



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

The University of Alberta

Declarative Semantics for Logic Programming

by

Yong Hu



A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Doctor of Philosophy

Department of Computing Science

Edmonton, Alberta
Fall 1994



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file *Votre référence*

Our file *Notre référence*

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-95195-8

Canada

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Yong Hu

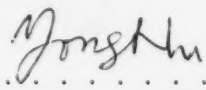
TITLE OF THESIS: Declarative Semantics for Logic Programming

DEGREE: Doctor of Philosophy

YEAR THIS DEGREE GRANTED: 1994

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Signed) 

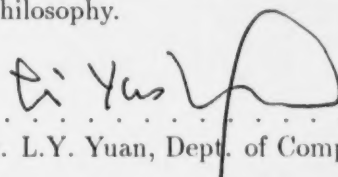
Permanent Address:
9-133 Woodridge Crescent,
Nepean, Ontario,
Canada, K2B 7T2

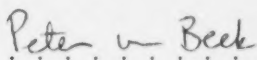
Date: June 24, 1994

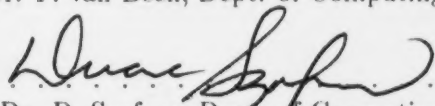
THE UNIVERSITY OF ALBERTA

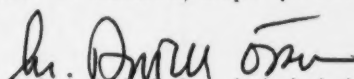
FACULTY OF GRADUATE STUDIES AND RESEARCH

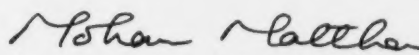
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Declarative Semantics for Logic Programming** submitted by **Yong Hu** in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

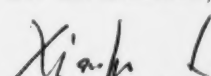

.....
Dr. L.Y. Yuan, Dept. of Computing Science
Supervisor


.....
Dr. P. van Beek, Dept. of Computing Science


.....
Dr. D. Szafron, Dept. of Computing Science


.....
Dr. M. Tamer Özsu, Dept. of Computing Science


.....
Dr. M. Matthen, Dept. of Philosophy


.....
Prof. P.H. Schmitt, Dept. of Computer Science
University of Karlsruhe, Germany

Date: June 24, 1994

To my parents

Abstract

Various declarative semantics for logic programming have been proposed, and their properties have been studied in several different formalisms. However, the different formalisms make it very difficult, if not impossible, to compare the semantics and to analyze their properties in a uniform way.

In this dissertation, we propose a new framework for the declarative semantics based on hypothetical reasoning. The four main issues addressed in the framework are the unified role of an underlying logic, the interpretation of negation **not** as failure, the rule-like behavior of implication $\leftarrow$ and the viewpoint of knowledge representation. An $\mathcal{L}$ -well-founded, an $\mathcal{L}$ -preferable or an $\mathcal{L}$ -stable semantics may be defined for an underlying logic $\mathcal{L}$ in the framework, corresponding to a skeptical, credulous or omniscient viewpoint of knowledge representation, respectively. In all the semantics, **not** l is uniformly interpreted as a hypothesis that l is justifiably assumed false while the rule-like behavior is represented by oriented consequences.

We study *disjunctive extended programs*, the ones that may contain both negation **not** as failure and classical negation $\neg$ and allow disjunctive heads, in the contexts of Lukasiewicz's m -valued logic L_m and autoepistemic logic $\mathcal{AE}$. We show that almost all existing semantics are equivalent to semantics derived from the newly proposed framework by choosing L_m or $\mathcal{AE}$ as the underlying logic and adopting a particular viewpoint of knowledge representation. With the framework, we are able to study existing semantics in the same formalism, and thus to provide a better understanding of the inherent relationship among them.

We also propose new semantics based on the framework since some semantics derivable from the framework have not been proposed. Surprisingly, they even include a new semantics for the nearly thoroughly studied *normal programs*, the ones that contain **not** but neither $\neg$ nor disjunctions, in addition to new semantics for disjunctive extended programs. Furthermore, the concept of *semantical instances*, first defined in this thesis for disjunctive programs, may also be used in other formalisms to extend semantics to disjunctive programs.

We postulate desirable criteria to classify semantics. One of the criteria is *the principle of locality*, which captures the "rule-like" behavior of $\leftarrow$. We also propose a

new approach to classify semantics using a criterion, called *cumulativity*, which has been identified as one of the most important properties for any bona fide nonmonotonic logic. Other criteria include *reflexivity*, *rationality*, *supraclassicality* and *the principle of proving by cases*.

Contents

1	Introduction	1
1.1	Major Contributions of the Dissertation	2
1.1.1	Factors of Defining Declarative Semantics	2
1.1.2	A Framework Based on Hypothetical Reasoning	5
1.1.3	Relationship with Abduction	6
1.1.4	Analysis of Existing Semantics	7
1.1.5	New Semantics Proposed	8
1.1.6	Criteria for Appropriate Semantics	11
1.2	Organization of the Document	12
2	Preliminaries	13
2.1	Multivalued First Order Logic	13
2.2	Herbrand Interpretations and Instantiations	20
2.3	Logic Programs	25
2.3.1	Syntax of Logic Programs	25
2.3.2	Partial Model Semantics	26
2.3.3	Queries and Answers	29
3	A Framework for Semantics of Logic Programming Based on Hypothetical Reasoning	31
3.1	A Simple Framework for Hypothetical Reasoning	32
3.2	A Framework for Semantics of Logic Programming	33
3.2.1	Incorporating Hypotheses	34
3.2.2	Noncontradictoriness	36
3.2.3	Positive Foundedness and False-Assumability	37
3.2.4	Admissibility	39
3.2.5	$\mathcal{L}$ -Saturated Extension Semantics	40
3.3	Fixed Point Characterization of $\mathcal{L}$ -Saturated Semantics	41
3.3.1	Lattice and Fixed Points	42

3.3.2	Fixed Point Characteristics of $\mathcal{L}$ -Saturated Hypothesis Tuples	46
3.4	A Framework for Semantics of Disjunctive Logic Programs	51
3.5	Semantical Extensions Based on $\mathcal{L}$ -Saturation	53
3.6	Impact of Classical Negation in Extended programs	54
3.6.1	Proof by Cases and Oriented Consequences	55
3.6.2	L_m -Saturated Answer Set Semantics for Extended Programs	58
3.7	Syntactical Characterization of $\mathcal{L}$ -Saturated Semantics	61
3.8	Appendix	65
4	Further Analyses: Normal Logic Program	70
4.1	Partial Model Representation of $\mathcal{L}$ -Saturated Semantics	70
4.2	Preferential Extension Semantics	72
4.3	The 3-valued Stable Model Semantics	75
4.4	The Well-founded Model Semantics	78
4.5	The Stable Model Semantics	79
4.6	Semantics Based on AE logic	80
4.6.1	Autoepistemic Logic	81
4.6.2	$\mathcal{AE}$ -Preferable Semantics	82
4.7	New Semantics	85
4.8	The Least Herbrand Model Semantics for Definite Programs	87
4.9	Summary	91
5	Further Analyses: Extended Logic Program	94
5.1	Answer Set Semantics	94
5.1.1	GL-Answer Set	94
5.1.2	Autoepistemic Representation of GL-Answer Set	96
5.2	Argumentation Semantics	98
5.3	New Semantics	100
6	Classification of Semantics	102
6.1	Introduction	102
6.2	The Principle of Locality	103
6.2.1	Relevant Set and Relevant Program	103
6.2.2	Principle of Locality	105
6.2.3	Principle of Setwise Locality	109
6.2.4	Applications of the Principles of Locality	110
6.3	Cumulative and Rational Monotony of Semantics	114
6.3.1	Kraus/Lehmann/Magidor's rules and Makinson's Operation	114
6.3.2	Cumulativity and Rationality	116

6.4	Supraclassicality	123
6.5	Principle of Proof by Cases	125
7	Related Work and Future Research	128
7.1	Related Work	128
7.2	Suggestions for Future Research	129
7.2.1	Procedural Semantics	129
7.2.2	Equivalence of Logic Programs	131
7.2.3	Update Semantics	131

Chapter 1

Introduction

Logic programming is an outgrowth of the so called "logical approach" to knowledge representation in artificial intelligence [54, 77]. It uses a first order logic, or at least a subset of it, as a programming language, and provides a mechanism for achieving automated deduction and, further, artificial intelligence [30]. The fields of logic programming and deductive databases are intimately related [59]. The close relationship between logic programming and nonmonotonic reasoning has also been established [39, 40, 68, 70, 72].

A main idea of logic programming is the separation of logic and control of a program. The logic concerns *what* the problem is that is to be solved. The control concerns *how* it is to be solved [42]. Such a separation of logic and control naturally results in the separation of the declarative semantics and the procedural semantics of a logic program. The declarative semantics specifies what conclusions are to be inferred from a program, but does not prescribe how. The procedural semantics is specified by a proof procedure as to how conclusions are to be derived from a program. One of the challenging tasks in the field of logic programming is to give a proper definition of the declarative semantics for logic programming.

A logic program is a set of sentences of the following form

$$L_0 \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n.$$

The study of logic programming has developed from definite programs where the L_i 's are atoms and $m = n$, to normal programs, which may contain **not** known as "negation as failure". It has further developed to extended programs, where the L_i 's are literals, i.e. they may be of the form $\neg a$ where $\neg$ is the ordinary or classical negation. Another dimension of the development is disjunctive programs, where L_0 is a disjunction of the form $l_1 \vee \dots \vee l_k$.

Various semantics have been proposed for different types of logic programs, and their properties have been analyzed under different formalisms. Recent attention

has been focused on extended programs, which contain both classical negation and negation as failure [6, 13, 40, 49]. However, the different formalisms of those semantics make it very difficult, if not impossible, to analyze and compare the semantics and to establish the relationship among the semantics.

A uniform framework for declarative semantics of logic programming is proposed in this dissertation. The framework unifies existing and newly proposed semantics of logic programming. Thus, it aids analysis and comparison of semantics, and facilitates the development of new semantics and the extension of existing semantics into the context of (disjunctive) extended programs from simpler forms of logic programs.

1.1 Major Contributions of the Dissertation

In this section, we summarize the major contributions of this dissertation.

1.1.1 Factors of Defining Declarative Semantics

Whether a semantics is the “right” one really depends on people’s expectations. After all, logic programs are tools whose behaviors need to be understandable and manageable by people. Three important factors contributing to people’s expectations of the behaviors of a logic program are the underlying logic, the meaning of negation as failure **not** and the interpretation of implication $\leftarrow$. In this dissertation, we analyze each one of the three factors and identify the roles they play in defining semantics of logic programs.

Underlying Logic: Logic programs are sets of sentences (in a certain form) in a language of a logic. Presumably, the logic has a semantics. In searching for a proper semantics for logic programming, the semantics of the logic is bound to play an important and fundamental role. No single logic is suitable for all applications of knowledge representation and reasoning. Some logics may be more suitable in certain aspects than others. For instance, the classical first order logic is appropriate to model deductive reasoning, and autoepistemic logic is appropriate to model the beliefs of an ideal rational agent [61]. Clearly, different semantics for logic programming would be defined because of the characteristics and nature of the underlying logics.

The role of the semantics of an underlying logic has not been presented in a uniform way in the literature. The logical consequence relation $\models_{\mathcal{L}}$ is used to reflect the semantics of any logic $\mathcal{L}$, in the course of developing the framework and proposing $\mathcal{L}$ based semantics of logic programming. Thus, we demonstrate

how the semantics of the underlying logic $\mathcal{L}$ plays a uniform and fundamental role in defining the semantics for logic programming. Specifically, we study Lukasiewiczian m -valued logic and autoepistemic logic as the underlying logic. These logics have well formulated semantics. It turns out that almost all existing semantics correspond to an $\mathcal{L}$ based semantics for some choice of $\mathcal{L}$. For instance, the well-founded model semantics [23] is equivalent to the $\mathcal{L}$ -well-founded semantics when $\mathcal{L}$ is Lukasiewiczian 3-valued logic while the stable model semantics [24] is equivalent to $\mathcal{L}$ -preferable semantics when $\mathcal{L}$ is the autoepistemic logic.

As a result, the major weakness of the well-founded model semantics, i.e. the inability to handle any conclusions reachable only by proof by cases, is shown to be caused by the same inability of the 3-valued logic. Thus in order to achieve the ability to prove by cases, the underlying logic chosen must be capable of proving by cases itself. Such logics include classical first order logic and the autoepistemic logic.

Negation as Failure not: Negative information is not to be exhaustively and explicitly expressed in most logic programs. Negative conclusions are to be reached under assumptions. The need for such assumptions arises from the fact that the description of the world given by a database of knowledge is incomplete, especially for descriptions about negative conclusions. For example, it is feasible for a database to describe people who are students of a university, but it is almost impossible (at least practically) for the database to describe people who are **not** students of the university. The negation **not** as failure is introduced to accommodate such assumptions. Most assumptions are unsound in the sense that conclusions are defeasible in light of new facts. How to interpret negation depends on what assumptions to adopt. For instance, under the closed world assumption (CWA) [75], p is inferred not true if p is not a logical consequence of a program in first order logic. CWA is the assumption adopted in the least Herbrand model semantics for definite programs [42]. In the well-founded model semantics, negative conclusions are inferred in terms of an *unfounded set* [23]. In the stable model semantics, negative literals are assumed to be negated belief atoms [24]. Different semantics have been achieved under different assumptions adopted in logic programming.

In this dissertation, **not** l is uniformly interpreted as a hypothesis that l is justifiably assumed false. Although **not** is interpreted uniformly, we are still able to propose various semantics by choosing different underlying logics and adopting different points of view of knowledge representation. It turns out that

various assumptions for negative conclusions adopted in the existing semantics can be unified as acceptable hypotheses characterized by their *admissibility*.

Rule-like Implication $\leftarrow$: The symbol $\leftarrow$ in logic programming is assumed to have “rule-like” behaviors, and is not to be treated as material implication. It bears some embedded meanings that the left hand side is to be inferred if the right hand side can be inferred or assumed. For instance, $a \leftarrow \neg b$ and $a \vee b$ are logically equivalent in the classical first order logic. In the context of logic programming, when $a \leftarrow \neg b$ is represented as $a \leftarrow \text{not} b$, a is to be inferred if b can be assumed false. Such a statement says nothing about the inferability of b while $a \vee b$ says that either a or b is true but does not specify which one. The latter statement does say something about the truth of b . Consider two extended programs $\Pi_1: \{b \leftarrow a; a \leftarrow\}$ and $\Pi_2: \{\neg a \leftarrow \neg b; a \leftarrow\}$. The two programs are logically equivalent in first order logic. However, both a and b are to be inferred from Π_1 while only a is to be inferred from Π_2 since there are no rules in Π_2 to infer b . Another important consideration about $\leftarrow$ is whether the “rule-like” implication $\leftarrow$ is allowed to be used in proving by cases, as indicated in the following program Π .

$$\begin{array}{ll} \Pi : & a \leftarrow \neg b. \\ & b \leftarrow \neg a. \\ & d \leftarrow a. \\ & d \leftarrow b. \end{array}$$

Is d to be inferred even if neither a nor b can be inferred from Π ? If $\leftarrow$ is allowed to be used in proving by cases, d can be inferred from Π . Otherwise, d can not be inferred. Allowing or not allowing proof by cases results in different interpretations of $\leftarrow$, and hence different semantics for logic programming.

We propose the concept of *oriented consequences*, and define two inference relations, $\vdash_{Horn}$ and $\vdash_{Case}$. $\vdash_{Horn}$ corresponds to the interpretation of $\leftarrow$ that disallows proof by cases while $\vdash_{Case}$ corresponds to the interpretation that allows proof by cases. Moreover, $\vdash_{Horn}$ and $\vdash_{Case}$ live up to the semantics of the 3-valued and 2-valued logic respectively to such an extent that the framework developed using $\models_{\mathcal{L}}$ is redefined using $\vdash_{Horn}$ and $\vdash_{Case}$, and almost all the properties of the framework are preserved. The syntactically defined $\vdash_{Horn}$ and $\vdash_{Case}$ also bridge the gap between the syntax of logic programs and the semantics of the underlying 3-valued or 2-valued logic. The introduction of oriented consequences is essential for the framework to make sense in the context of extended programs, which contain both negation **not** as failure and classical negation $\neg$.

1.1.2 A Framework Based on Hypothetical Reasoning

Based on an analysis of the three major factors, we formalize the declarative semantics of logic programming under a framework, which is based on hypothetical reasoning, analyze properties of the semantics, and compare various semantics. We demonstrate in the framework that an underlying logic plays a unifying role in defining declarative semantics. Negation **not** as failure is interpreted as acceptable hypotheses. In our framework, hypotheses are expressed in terms of a tuple $\langle P, N \rangle$, where P is a set of positively assumed hypotheses while N is a set of negatively assumed hypotheses. The basic idea for a hypothesis in N to be acceptable is simple and very effective, and was first proposed by Dung [12]. It says that h is acceptable if all hypotheses that reject h are to be rejected. A thesis we adopt in our research is that positive hypotheses are to be minimized while negative hypotheses are to be maximized. Thus, the idea for a hypothesis h in P to be acceptable is that h is acceptable if h has to be assumed. Acceptable hypothesis tuples are characterized by their *admissibility* — a concept that describes the acceptability of positive and negative hypotheses and other properties of a tuple. We shall study structural properties of the set of all admissible tuples, and show that it constitutes a complete semilattice.

Guided by the same thesis of maximizing negative hypotheses and minimizing positive hypotheses, we define the *saturated closure* of an admissible tuple, which is maximized with acceptable hypotheses and preserves the admissibility. Not only does the set of saturated tuples constitute a semilattice, each saturated tuple is also characterized as a fixed point of a monotonic operator defined on the semilattice of admissible tuples. Various semantics are proposed based on saturated hypothesis tuples for an underlying logic $\mathcal{L}$. *$\mathcal{L}$ -well-founded semantics*, characterized as the least fixed point of the operator, represents a skeptical point of view of knowledge representation; *$\mathcal{L}$ -preferable semantics*, characterized as maximal fixed points of the operator, represents a credulous point of view; and *$\mathcal{L}$ -stable semantics* corresponds to an omniscient point of view that in every possible world represented by a knowledge base, everything is either true or false. Both *$\mathcal{L}$ -well-founded semantics* and *$\mathcal{L}$ -preferable semantics* are defined for the class of all programs while *$\mathcal{L}$ -stable semantics* is defined only for a subset of programs. Besides being characterized as fixed points, saturated tuples will also be characterized syntactically with $\vdash_{Horn}$ and $\vdash_{Case}$, and thus various saturated semantics are also justified syntactically.

Our framework is defined for a quite general context of logic programming; it is applicable to any disjunctive extended programs, which contain both **not** and $\neg$ and allow disjunctive heads. Consequently, the framework provides a uniform foundation to analyze properties of various semantics and compare almost all existing semantics. As a result, the framework provides a good understanding of the inherent relationship

among various semantics. Each existing semantics for logic programming proposed in the literature is defined based on a certain underlying logic with particular interpretations of negation and implication. In other words, each semantics is presented in a somewhat ad hoc formalism. To the best of our knowledge, there has not been a unified view on this issue, and the relationship among various semantics has not been established in a uniform way although the relationship between the two most prominent semantics — the well-founded and the stable model semantics — has been established [23, 73]. Our framework provides a unified view on this issue since it turns out that each of the existing semantics we studied is equivalent to one of the saturated semantics for a specific choice of underlying logic and a particular point of view of knowledge representation.

In addition to the unifying role of the framework, the concept of *semantical instances*, defined in the framework for disjunctive programs, can be easily extended beyond the framework, and be used in other formalisms to define semantics for disjunctive programs.

Our framework significantly extends the framework proposed by Dung [12]. His framework is only applicable to normal programs, which contain neither classical negation $\neg$ nor disjunction, while ours is applicable to programs that may contain both classical negation and disjunction. Positive hypotheses are not studied in his work. It turns out that positive hypotheses play a crucial role in extending our framework to the context of disjunctive programs. The role of an underlying logic is not studied either in his work. Although his framework is defined in classical logic, we shall show that it is actually equivalent to a special case of our framework when applied only to normal programs with the underlying logic being the 3-valued logic.

1.1.3 Relationship with Abduction

The relationship between logic programming and nonmonotonic reasoning has drawn a lot of attention. An enormous amount of work has been done on relating logic programming with the four major formalisms for nonmonotonic reasoning — McCarthy's circumscription [52, 53], Reiter's default theory [76], Moore's autoepistemic logic [61], Reiter's closed world assumption [75] and its extension, Minker's generalized closed world assumption (GCWA) [58]. However, the relationship between logic programming and abduction, which is also a form of nonmonotonic reasoning because hypotheses may become unacceptable in light of new facts, has received much less attention.

The term "abduction" coined by the philosopher Charles Peirce refers to a particular kind of hypothetical reasoning — inference to the best explanations [16]. In the simplest case, it has the form:

From α and $\alpha \leftarrow \beta$

infer β as a possible "explanation" of α .

Although among the three basic forms of inference — deduction, induction and abduction, abduction is the least understood and has received relatively less attention, it has found applications in many areas of AI [66, 41], including the area of logic programming. Eshghi and Kowalski are probably the first to study the semantics for logic programming from a point of view of hypothetical reasoning, and relate the stable model semantics with abduction [15]. Dung has made significant progress in this direction by relating the well-founded model semantics with abduction [12] and proposing preferential extension semantics. In this dissertation, we also contribute significantly in establishing the relationship between logic programming and abduction since our framework is based on hypothetical reasoning. Our work is much more uniform and general than their work because it is applicable to more semantics.

1.1.4 Analysis of Existing Semantics

We shall show that almost all well known semantics fit into our framework. Let L_3 and L_2 denote the Lukasiewiczian 3-valued and 2-valued logic respectively, and $\mathcal{AE}$ denote the autoepistemic logic. Then

1. The well-founded semantics for normal programs [23] is equivalent to L_3 -well-founded semantics (Theorem 4.12);
2. The well-founded-by-case semantics for normal programs [80] is equivalent to L_2 -well-founded semantics (Theorem 4.12);
3. The 3-valued-stable model semantics for disjunctive normal programs [73] is equivalent to L_3 -saturated semantics (Theorem 4.10);
4. The stable model semantics for normal programs [24] is equivalent to $\mathcal{AE}$ -preferable semantics and L_3 -stable semantics (Theorem 4.16, 4.14);
5. The stable-by-case semantics for normal programs [80] is equivalent to L_2 -stable semantics (Theorem 4.14);
6. The regular model semantics for normal programs [93] is equivalent to L_3 -preferable semantics (Theorem 4.3 and [95]);
7. The preferential extension semantics for normal programs [12] is equivalent to L_3 -preferable semantics (Theorem 4.3);

8. The argumentation semantics for extended programs [13] is extended by L_3 -preferable answer set semantics (Theorem 5.9);
9. The answer set semantics for disjunctive extended programs [40] is equivalent to L_3 -stable answer set semantics (Theorem 5.3).

All the semantics defined in the framework are based on saturated tuples. A saturated tuple is considered a description of a possible world represented by a logic program. For such a tuple $\langle P, N \rangle$, P is the set of positive conclusions and N is the set of negative conclusions reachable in the possible world. The $\mathcal{L}$ -well-founded semantics is defined as the smallest saturated tuples. These tuples contain only conclusions that are true in every possible world. Thus, the semantics is a skeptical semantics. The $\mathcal{L}$ -preferable semantics is defined as maximal saturated tuples. Such a tuple contains as many conclusions reachable in a possible world as possible (under the condition of admissibility). Thus, the semantics is a credulous semantics. These observations are also supported by the fixed point theory discussed before since all $\mathcal{L}$ -saturated tuple are characterized as fixed points of a monotonic operator. The $\mathcal{L}$ -stable semantics is defined as tuples $\langle P, N \rangle$ such that P and N partition the set of all conclusions. For such a tuple $\langle P, N \rangle$, every conclusion is either true in the possible world and contained in P , or false and contained in N . Thus, the semantics is an omniscient semantics.

The above list indicates that each of the existing semantics listed corresponds to a semantics proposed in the framework for a particular choice of an underlying logic among L_3 , L_2 and $\mathcal{AE}$, and an adoption of a particular point of view of knowledge representation — skepticism for the well-founded semantics, credulity for the preferable semantics and omniscience for the stable semantics.

We do not discuss the predicate completion semantics [7] and the 3-valued predicate completion [18, 33] because the interpretation of $\leftarrow$ is $\leftrightarrow$ (if-and-only-if) as applied in the predicate completion semantics. Such an interpretation of $\leftarrow$ is fundamentally different from the rule-like interpretation. Properties and some bizarre behaviors of predicate completion have been studied by Shepherdson [82, 83]. It has been shown that the well-founded model semantics extends Fitting and Kunen's predicate completion based on 3-valued logic [23].

1.1.5 New Semantics Proposed

As indicated by the last section, some of the semantics derived from our framework are equivalent to some existing semantics. Since the existing semantics are defined only for a particular type of logic programs while our semantics are defined for disjunctive extended programs, we extend the semantics to a broader scope of logic programs.

The idea is based on the concept of *semantical instances* defined in the framework for disjunctive programs. The concept is first defined in this dissertation, and is so general that it can be used in other formalisms. Furthermore, we also propose new semantics. We summarize them as follows.

1. L_2 -preferable semantics is a new semantics for both normal programs and disjunctive normal programs;
2. L_m -saturated answer set semantics is a new semantics for disjunctive extended programs, where $m = 2, 3$;
3. L_m -preferable answer set semantics is a new semantics for disjunctive extended programs, where $m = 2, 3$;
4. L_2 -well-founded semantics extends the well-founded-by-case model semantics to disjunctive normal programs;
5. L_2 -stable semantics extends the stable-by-case model semantics to disjunctive normal programs;
6. L_3 -preferable semantics extends the preferential extension semantics to disjunctive normal programs;

L_2 -preferable semantics is a credulous semantics for (disjunctive) normal programs which is capable of proving by cases while L_m -saturated answer set semantics is a skeptical semantics for (disjunctive) extended programs. Although the preferred extension semantics [13] is a credulous semantics for extended programs, the L_3 -preferable answer set semantics is even more "credulous" in that it would give more answers. Moreover, L_m -preferable semantics are defined for disjunctive extended programs.

The last three existing semantics in the above list were originally defined only for normal programs. It appeared that there was no trivial way to extend the semantics to disjunctive programs using the same approaches. As indicated by the above list, our framework also provides a useful approach to propose new semantics and to extend existing semantics as well.

The following tables show the relationships among different semantics for various types of logic programs. The existing semantics listed in each cell are equivalent to the $\mathcal{L}$ based semantics in the same row with $\mathcal{L}$ being instantiated with the logic in the same column. An x means that the semantics of the logic in the same column is not compatible with the viewpoint listed in the same row.

Semantics for Normal Programs

$\mathcal{L}$	L_3	L_2	$\mathcal{AE}$
$\mathcal{L}$ -well-founded (Skeptical View)	The well-founded model	The well-founded -by-case model	x
$\mathcal{L}$ -saturated (Skeptical View)	3-valued stable models	The well-founded -by-case model	x
$\mathcal{L}$ -preferable (Credulous View)	Regular model Preferential extension	NEW	The stable model
$\mathcal{L}$ -stable (Omniscient View)	The stable model	The stable -by-case	The stable model

Semantics for Disjunctive Programs

$\mathcal{L}$	L_3	L_2	$\mathcal{AE}$
$\mathcal{L}$ -well-founded (Skeptical View)	3-valued stable model	NEW	x
$\mathcal{L}$ -saturated (Skeptical View)	3-valued stable models	NEW	x
$\mathcal{L}$ -preferable (Credulous View)	NEW	NEW	GL-answer set
$\mathcal{L}$ -stable (Omniscient View)	GL-answer set	NEW	GL-answer set

Semantics for Extended Programs

$\mathcal{L}$	L_3	L_2	$\mathcal{AE}$
$\mathcal{L}$ -well-founded (Skeptical View)	NEW	NEW	x
$\mathcal{L}$ -saturated (Skeptical View)	NEW	NEW	x
$\mathcal{L}$ -preferable (Credulous View)	Extend Argumentation Semantics	NEW	GL-answer set
$\mathcal{L}$ -stable (Omniscient View)	GL-answer set	NEW	GL-answer set

Semantics for Disjunctive Extended Programs

$\mathcal{L}$	L_3	L_2	$\mathcal{AE}$
$\mathcal{L}$ -well-founded (Skeptical View)	NEW	NEW	x
$\mathcal{L}$ -saturated (Skeptical View)	NEW	NEW	x
$\mathcal{L}$ -preferable (Credulous View)	NEW	NEW	GL-answer set
$\mathcal{L}$ -stable (Omniscient View)	GL-answer set	NEW	GL-answer set

1.1.6 Criteria for Appropriate Semantics

It seems fair to say that each well defined semantics has its advantages and weaknesses in comparison with other semantics. In order to analyze various semantics, criteria are needed to classify them. In this dissertation, we postulate criteria of appropriate semantics. Such criteria include:

Principle of Locality: The “rule-like” behavior assumed for the implication symbol $\leftarrow$ has been discussed and presented only very informally in the literature. We postulate the *principle of locality* and formulate the notion of “rule-like” behavior of $\leftarrow$. Roughly, the principle of locality says that the meaning of a literal is to be determined only by clauses which may be used to derive the literal.

Cumulativity: Research has been done to postulate desirable properties of logics for nonmonotonic reasoning [44, 32]. Among them is the most important one – *cumulativity*. Intuitively, cumulativity says that adding a set of formulae inferable from a set of premises to the set of premises does not change the set of all formulae inferable from the premises. Because of the intimate relationship between logic programming and nonmonotonic reasoning, it would be a natural outcome of the research to study semantics for logic programming in terms of the desirable properties postulated for logics of nonmonotonic reasoning. Dix is the first to use the properties to classify semantics for logic programming [8]. We formulate these properties in the setting suitable to logic programming and compare our work with Dix’s work.

Principle of Proof by Cases: As a result of study on the properties of the non-monotonic but cumulative consequence relation $|\sim$, Makinson suggested the

following rule, which expresses the principle of proof by cases [44].

Proof by Cases:

$$\frac{\alpha \wedge \neg \beta \mid \sim \gamma, \quad \alpha \wedge \beta \mid \sim \gamma}{\alpha \mid \sim \gamma}$$

We also formulate the principle for the semantics of logic programming.

1.2 Organization of the Document

Chapter 1 summarizes motivations and major contributions of this dissertation.

Chapter 2 introduces the syntax and terminology of logic programming. Although we adopt the terminology introduced in Lloyd's book [42], the notions and concepts are defined in terms of Lukasiewicz's many-valued logic, instead of the conventional classical logic. The requisite concepts of the logic such as languages, interpretations, models and logical consequences are also covered in this chapter.

Chapter 3 develops a general theory based on hypothetical reasoning as a framework of semantics for logic programming. The properties of the framework are studied, including its fixed point characteristics and syntactical characteristics. The framework is defined for a broad scope of logic programming, and is applicable to programs that may contain both negation **not** as failure and classical negation $\neg$ and allow disjunctive heads.

Chapter 4 shows most existing prominent semantics for (disjunctive) normal programs fit in the framework in that they are equivalent to a semantics derived from the framework. New semantics for (disjunctive) normal programs are also proposed in this chapter.

Chapter 5 shows most existing prominent semantics for (disjunctive) extended programs fit in the framework in that they are equivalent or correspondent to a semantics derived from the framework. New semantics are also proposed for (disjunctive) extended programs in this chapter.

Chapter 6 postulates some criteria to classify various semantics. The principle of locality aims at measuring the "rule-like" behavior of the implication symbol $\leftarrow$ represented in a semantics. This principle also says that the intuitive meaning of the orientation of $\leftarrow$ should be represented by a proper semantics. Desirable properties postulated for nonmonotonic logic are formulated so that they are suitable for semantics of logic programs. Among these properties are *cumulativity*, *supraclassicality* and *principle of proof by cases*.

Chapter 7 concludes by discussing related work and suggesting future research.

Chapter 2

Preliminaries

2.1 Multivalued First Order Logic

This section introduces some requisite concepts of a multivalued first order logic, including languages, sentences of a language, interpretations of a language and models for a set of sentences.

We now proceed to define alphabets and first order languages.

Definition 2.1 An *alphabet* consists of the following classes of symbols: variables, n -ary functions, n -ary predicates, connectives, quantifiers and truthvalues

We assume the set of variables is a countably infinite set, and that indeed there is some given enumeration of it (without repetition) by reference to which we can unambiguously refer to the j th individual variable, where j is any positive integer. Variables will be normally denoted by letters u, v, w, x, y and z (possibly subscripted). 0-ary functions are called *constants*. Constants, which may be taken as a class of symbols, will normally be denoted by letters a, b and c (possibly subscripted). Functions of various arities > 0 will normally be denoted by the letters f, g and h (possibly subscripted). Predicates of various arities > 0 will normally be denoted by the letters p, q and r (possibly subscripted). Occasionally, such conventions will not be applied rigorously for the sake of convenience. Possible confusion will be avoided from the context. The connectives are $\neg, \wedge, \vee, \rightarrow$ and $\leftrightarrow$, while the quantifiers are $\forall$ and $\exists$. The truthvalues are a set of entities that sentences, or closed formulae (defined later), of the logic stand for, or “denote”. There is another class of symbols called punctuation symbols. Punctuation symbols are “(”, “)” and “,”. These symbols are not symbols of the alphabet for a logic, but symbols used for better presentation and readability on paper.

Next we turn to the definition of the first order language given by an alphabet.

Definition 2.2 A *term* is defined inductively as follows:

- (a) A variable is a term;
- (b) If f is an n -ary function and $t_1, \dots, t_n$ are terms, then $f(t_1, \dots, t_n)$ is a term.

Definition 2.3 An *atomic formula*, or an *atom* for short, is an expression of the form $p(t_1, \dots, t_n)$, where p is an n -ary predicate and $t_1, \dots, t_n$ are terms. A (*well-formed*) *formula* is defined inductively as follows:

- (a) a truthvalue is a formula;
- (b) an atomic formula is a formula;
- (c) If F and G are formulae, so are $(\neg F)$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$ and $(F \leftrightarrow G)$;
- (d) If F is a formula and x is a variable, so are $\forall x F$ and $\exists x F$.

It will often be convenient to write the formula $(F \rightarrow G)$ as $(F \leftarrow G)$.

Definition 2.4 The *first order language* given by an alphabet consists of the set of all formulae constructed from the symbols of the alphabet.

A first order language is abbreviated to a language hereafter.

We defined two classes of expressions — terms and formulae. The general idea of first order predicate calculus is, informally, that there is a set of things, which are “what the formulae are about”. These things are called the *individuals*, and the set containing all of them is called the *universe of individuals*, or *universe* for short. The intended meaning of terms is that they stand for, or “denote”, the individuals. On the other hand, formulae are intended to express propositions about the individuals. These propositions may have different truth status. The conventional, probably prevailing, view was that every proposition is either true or else false — although which of these is the case may well neither be *necessary* as regards the matter itself nor *determinable* as regards our knowledge of it. Propositions about future contingents, like that asserting the occurrence of a sea battle tomorrow, are neither actually true nor actually false, but potentially either, and thus have a third, indeterminate truth status. Discussions of future contingency provide the leading idea and principal impetus for the development of Lukasiewicz’s 3-valued logic [78, p. 3]. The following example from [18] illustrates the idea behind Kleene’s 3-valued logic. Say P is a statement asserting $f(3) = 5$, where f is a function we have a Turing machine to compute. By running the Turing machine we may determine the truth or falsity of P , or we may never learn anything if the machine fails to halt. So a third truth value represents

the undetermined or undefined situation. Important sources of inspiration for many valued logic also include propositional modality, such as the so called *althic* modalities¹, and *probabilistic* modalities² [78, p 3]. See [78, Chapter 1] for a historical conspectus of many-valued logic.

In the predicate calculus, the truth status are represented by distinct truth values. To say that a formula expresses a *true* proposition is to say that the formula stands for, or “denotes”, *truth*, which is considered a truthvalue. Similarly, a formula, which expresses a *false* proposition, is said to stand for, or denote, the truthvalue *falsity*. Furthermore, the truth or falsity of a proposition expressed by a formula may well not be determinable. In the case of Kleene’s logic, the formula is said to denote *undefinedness*, or *indeterminateness*, which is also considered a truthvalue. For different universes and propositions with different nature, a first order logic would need different sets of truthvalues. We are going to return to this issue when we discuss the semantics of first order logics later. For the time being, the set of truthvalues of an alphabet is considered a set of entities, and all formulae given by an alphabet denote these entities.

The informal meaning of connectives and quantifiers is as follows. $\neg$ is negation, $\wedge$ is conjunction, $\vee$ is disjunction, $\rightarrow$ is implication and $\leftrightarrow$ is equivalence. $\forall$ is the universal quantifier, while $\exists$ is the existential quantifier. Thus “ $\forall x$ ” means “for all x ” and “ $\exists x$ ” means “there exists an x ”.

Definition 2.5 An occurrence of a variable x in a formula F is *bound* in F if it occurs in a part of F of the form $\forall xG$ or the form $\exists xG$; otherwise, it is *free* in F . x is said to be *free* (*bound*) in F if some occurrence of x is free (bound) in F .

It should be noted that a variable may be both free and bound in a formula.

Definition 2.6 A *sentence*, which is also called a *closed formula*, is a formula with no free occurrences of any variables.

Definition 2.7 If F is a formula, then the *universal closure* of F , denoted as $\forall(F)$, is a sentence obtained by adding a universal quantifier for every free variable of F . Similarly, the *existential closure* of F , denoted as $\exists(F)$, is a sentence obtained by adding an existential quantifier for every free variable of F .

Definition 2.8 A *literal* is an atom or the negation of an atom. A *positive literal* is an atom. A *negative literal* is the negation of atom.

¹1. necessarily true; 2. contingently (i.e., actually but not necessarily) true; 3. contingently false; 4. necessarily false.

²1. certainly true; 2. probably true; 3. indifferent; 4. probably false; 5. certainly false.

Now we proceed to introduce the semantics of a multivalued first order logic. First of all, we define interpretations of a language and models of sentences.

Definition 2.9 An *interpretation* of a first order language $\mathbf{L}$ consists of the following:

- (a) A nonempty set U , called the *universe* of the interpretation;
- (b) For each 0-ary function, or constant, in $\mathbf{L}$, the assignment of an individual in U ;
- (c) For each n -ary function, $n > 0$, in $\mathbf{L}$, the assignment of a mapping from U^n to U ;
- (d) For each n -ary predicate in $\mathbf{L}$, the assignment of a mapping from U^n to V , where V is the set of truthvalues in $\mathbf{L}$.

As indicated informally in the last section, a term is intended to denote an individual in a universe and a formula is intended to denote a truthvalue. Consider an n -ary predicate p and n terms $t_1, \dots, t_n$. Suppose p expresses a certain proposition and $t_1, \dots, t_n$ denote individuals $i_1, \dots, i_n$. Then $p(t_1, \dots, t_n)$ expresses the proposition about $i_1, \dots, i_n$. Corresponding to different truth status of the proposition, $p(t_1, \dots, t_n)$ denotes different truthvalues. However, a term or a formula with free variables does not denote any particular thing since variables range over the universe and with different assignments of individuals to the variables, the term or the formula may denote different individual or truthvalue.

Definition 2.10 Let I be an interpretation of a language $\mathbf{L}$. A *variable assignment wrt I* is an assignment to each variable in $\mathbf{L}$ of an individual in the universe of I .

Definition 2.11 Let I be an interpretation with universe U of a language $\mathbf{L}$ and A a variable assignment wrt I , the individual in U denoted by a term in $\mathbf{L}$ can be determined as follows:

- (a) Each variable denotes the individual assigned to it according to A ;
- (b) Each constant denotes the individual assigned to it according to I ;
- (c) For an n -ary function f , if f' is the mapping assigned to f according to I and $t_1, \dots, t_n$ denote $t'_1, \dots, t'_n$, then $f(t_1, \dots, t_n)$ denotes the individual $f'(t'_1, \dots, t'_n) \in U$.

For any term t in $\mathbf{L}$, if t denotes t' , we write

$$/t/_{I,A} = t'$$

With a given interpretation and a variable assignment, the truthvalue denoted by an atomic formula is determined when the denotation of terms is given by the above definition. All other (complex) formulae are constructed with connectives and quantifiers from atomic formulae. In the classical first order logic, the set of truth values consists of **t** and **f**. **t** is intended to mean "truth" while **f** is intended to mean "falsity". Next we introduce the many-valued logic of Lukasiewicz, which is a generalization of his 3-valued logic [78]. In the logic, the truthvalues are represented by real numbers from the interval 0 to 1 (with 0 playing the role of **f** and 1 as **t**). Let us now divide the interval from 0 to 1 by inserting evenly spaced division points, for a total of m points ($m \geq 2$), then these m numbers comprise the truthvalue set V_m of an Lukasiewiczian m -valued logic L_m , i.e., $V_m = \{0, \frac{1}{m-1}, \dots, \frac{m-2}{m-1}, 1\}$.

Definition 2.12 Let I be an interpretation with universe U of a language L with truthvalue set V_m and A a variable assignment wrt I , the truthvalue $/F/_{I,A} \in V_m$ denoted by a formula F in L is given as follows:

- (a) $/v/_{I,A} = v$, for each $v \in V_m$;
- (b) For an n -ary predicate p and n terms $t_1, \dots, t_n$, if $p' : U^n \mapsto V_m$ is the mapping assigned to p according to I , then

$$/p(t_1, \dots, t_n)/_{I,A} = p'(t'_1, \dots, t'_n),$$

where for each i , $t'_i = /t_i/_{I,A}$;

- (c) If the formula is of the form $\neg F$, $F \wedge G$, $F \vee G$, $F \leftarrow G$ and $F \leftrightarrow G$, then the truth value denoted by the formula is given by the following equations:

$$/\neg F/_{I,A} = 1 - /F/_{I,A},$$

$$/F \wedge G/_{I,A} = \min\{/F/_{I,A}, /G/_{I,A}\},$$

$$/F \vee G/_{I,A} = \max\{/F/_{I,A}, /G/_{I,A}\},$$

$$/F \leftarrow G/_{I,A} = \begin{cases} 1, & \text{if } /F/_{I,A} \geq /G/_{I,A} \\ 1 - /G/_{I,A} + /F/_{I,A}, & \text{otherwise} \end{cases}$$

$$/F \leftrightarrow G/_{I,A} = /(F \leftarrow G) \wedge (G \leftarrow F)/_{I,A}$$

- (d) If the formula has the form $\forall x F$ or $\exists x F$, then the truth value denoted by the formula is given by the following equations:

$$/\forall x F/_{I,A} = \min\{/F/_{I,A(x/i)} : i \in U\},$$

$$/\exists x F/_{I,A} = \max\{/F/_{I,A(x/i)} : i \in U\},$$

where $A(x/i)$ is A except that x is assigned the individual $i \in U$.

Let us consider two particular first order languages and logics. The first language is the classical first order language with the truth values **t** and **f**. Such a language is also called a *2-valued* language and a logic defined on it is also called a *2-valued* first order logic. It is readily seen that L_2 is identical to the classical first order logic if we identify 1 with **t** and 0 with **f**. It is a straightforward task to check that the truth value given by the above definition for a formula is exactly the same truth value given in the classical first order logic [57]. For instance, $F \leftarrow G$ has the same truth value as $F \vee \neg G$, which may taken as the definition of $\leftarrow$. $\forall x F$ is true in the classical logic if for all x assigned an individual in the universe, F has the truth value **t**. In such case, $\min\{ /F /_{I,A(x/i)} : i \in U \} = \mathbf{t}$.

The second is a language $\mathbf{L}$ with the truthvalue set $V = \{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$. Such a language is called a *3-valued* language and a logic defined on it is called a *3-valued* first order logic. L_3 is the 3-valued logic when we identify $\{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$ with $\{0, \frac{1}{2}, 1\}$. Another 3-valued logic was introduced by Przymusiński [71]. The 3-valued logic is defined the same as L_3 as in Definition 2.12 except that

$$/F \leftarrow G /_{I,A} = \begin{cases} \mathbf{t} , & \text{if } /F /_{I,A} \geq /G /_{I,A} \\ \mathbf{f} , & \text{otherwise} \end{cases}$$

We denote this logic by $\mathcal{PL}$. In these 3-valued logics, $F \leftarrow G$ does not always denote the same truth value as $F \vee \neg G$ does. When both F and G denote **u**, $F \leftarrow G$ denotes **t** while $F \vee \neg G$ denotes **u**.

Given a many-valued logic system, we can classify certain of its truth values as *designated* — i.e., as representing “truth-like” truthvalues. In the classical logic, the only designated truth value is **t**. A sentence which uniformly denotes a designated truthvalue in every interpretation is a tautology of the many-valued logic — subject to the specified designation of truthvalues. Similarly, we can also classify certain of its truth values as *antidesignated* — i.e., as representing “false-like” truthvalues. In the classical logic, the only antidesignated truth value is **f**. A sentence which uniformly denotes an antidesignated truthvalue in every interpretation is a contradiction of the many-valued logic — subject to the specified designation of truthvalues.

From now on, we assume that in each truth value set, there is only one truth-value classified as designated, which is denoted by **t**, and there is only one truth value classified as antidesignated, which is denoted by **f**. All other truth values are *nondesignated*. For Łukasiewiczian many-valued logic, **t** is 1 and **f** is 0. Since any finite truthvalue set can be identified with V_m for some $m \geq 2$, we assume that each language has V_m as its truthvalue set for some $m \geq 2$ and the logic defined on the language is the Łukasiewiczian m -valued logic L_m unless otherwise indicated.

Definition 2.13 Let I be an interpretation with universe U of a language $\mathbf{L}$. I is said to be a *2-valued* interpretation if for every n -ary predicate p of $\mathbf{L}$, the mapping p' assigned to p satisfies the condition that $p'(i_1, \dots, i_n)$ is either **f** or **t**, for any $(i_1, \dots, i_n) \in U^n$.

In the above definition, the language $\mathbf{L}$ may have more than two truthvalues. However, in a 2-valued interpretation of $\mathbf{L}$, each formula denotes either **f** or **t** for any variable assignment.

With the semantics of connectives and quantifiers defined as in Definition 2.12 for Lukasiewiczian many-valued logic, most of the requisite concepts for a logic such as *models*, *logical consequences*, etc. can be defined the same way as in the classical first order logic. And some of the relationships between these concepts also hold. We proceed to define these concepts.

Obviously from the above definition, the truth value denoted by a sentence, or closed formula, does not depend on the variable assignment. Consequently, we can speak unambiguously of the truth value of a sentence with respect to an interpretation. We write $/F/I$ to denote the truth value of a sentence F in the interpretation I as defined in Definition 2.12.

Definition 2.14 Let I be an interpretation of a language $\mathbf{L}$, F a sentence and S a set of sentences in $\mathbf{L}$. I is a *model* for F if F denotes the truth value **t**, i.e. $/F/I = \mathbf{t}$. And I is a *model* for S if I is a model for each sentence in S .

Proposition 2.1 Let S be a set of sentences of the form $\forall(F \leftarrow G)$ in a 3-valued language $\mathbf{L}$ with truth values $\{0, \frac{1}{2}, 1\}$. An interpretation of $\mathbf{L}$ is a model for S in the Lukasiewiczian 3-valued logic L_3 if and only if it is a model for S in the Przymusiński's 3-valued logic $\mathcal{PL}$.

Proof: It directly follows from Definition 2.12 and the definition of Przymusiński's logic after Definition 2.12. $\square$

Definition 2.15 Let S be a set of sentences in a language $\mathbf{L}$. S is said to be *satisfiable* if $\mathbf{L}$ has an interpretation which is a model for S . S is said to be *valid* if every interpretation $\mathbf{L}$ is a model for S . S is said to be *unsatisfiable* if none of the interpretation of $\mathbf{L}$ is a model for S .

We also say that a satisfiable set S of sentences is *consistent*, and that an unsatisfiable set S is *inconsistent*. Although consistency and inconsistency of a set of sentences are syntactical concepts, such terminology would not cause any confusion since we do not introduce any concepts about the (syntactical) derivation or deduction of first order logic in this dissertation.

Example 2.1 The sentence $p \leftrightarrow \neg p$, where p is a 0-ary predicate of a 3-valued language, is satisfiable because there is an interpretation I such that $/p/I = \mathbf{u}$. Thus I is a model of the sentence. However, if p is a predicate of a 2-valued language, the sentence $p \leftrightarrow \neg p$ is unsatisfiable because for any interpretation, the truth value of the sentence is $\mathbf{f}$. Similarly, $p \vee \neg p$ is valid if p is in a 2-valued language, but it is not otherwise.

Different alphabets generate different languages, and different logics may be defined on the same language. Generically, we use $\mathcal{L}$ to denote a first order logic defined on a language $\mathbf{L}$. $\mathcal{L}$ is a Lukasiewiczian many-valued logic unless otherwise indicated. We also say that a consistent (satisfiable, unsatisfiable or valid) set of sentences in a language $\mathbf{L}$ is consistent (satisfiable, unsatisfiable or valid) in logic $\mathcal{L}$.

We now give the definition of an important concept of logical consequence.

Definition 2.16 Let S be a set of sentences and F a sentence in a language $\mathbf{L}$. F is said to be a logical consequence of S if, for every interpretation I of $\mathbf{L}$, I is a model for S implies that I is a model for F .

For a set S of sentences and a sentence F , we write $S \models_{\mathcal{L}} F$ to mean that F is a logical consequence of S in logic $\mathcal{L}$, and $\models_{\mathcal{L}} F$ to mean that F is valid in $\mathcal{L}$. The subscript $\mathcal{L}$ in $\models_{\mathcal{L}}$ is necessary because different logic may have different semantics. Therefore, the logical consequence relation $\models_{\mathcal{L}}$ is different as indicated in the next example.

Example 2.2 Let S be $\{b \leftarrow a, b \leftarrow \neg a\}$. If S is a set of sentences of a 2-valued language, then $S \models_{\mathbf{L}_2} b$. However, if S is a set of sentence of a 3-valued language, $S \not\models_{\mathbf{L}_3} b$ because any interpretation I satisfying that $/a/I = /b/I = \mathbf{u}$ is a model for S but not a model for b .

Note that for any $m \geq 2$, $S \models_{\mathbf{L}_m} F$ if $\models_{\mathbf{L}_m} (F_1 \wedge \dots \wedge F_n) \rightarrow F$ when $S = \{F_1, \dots, F_n\}$ is a finite set of sentences. But the reverse is true only when $m = 2$.

2.2 Herbrand Interpretations and Instantiations

We now proceed to study a very important class of interpretations, i.e. the so-called Herbrand interpretations.

Definition 2.17 A *ground* term is a term containing no variables. A *ground* atom is an atom containing no variables.

Definition 2.18 Let $\mathbf{L}$ be a first order language. The *Herbrand universe* $U_{\mathbf{L}}$ for $\mathbf{L}$ is the set of all ground terms, which can be formed out of the constants and functions appearing in $\mathbf{L}$. (If $\mathbf{L}$ has no constants, we add some constant, say, a , to form ground terms).

Definition 2.19 Let $\mathbf{L}$ be a first order language. The *Herbrand base* $B_{\mathbf{L}}$ for $\mathbf{L}$ is the set of all ground atoms which can be formed by using predicates from $\mathbf{L}$ with ground terms from the Herbrand universe $U_{\mathbf{L}}$ as arguments.

Definition 2.20 Let $\mathbf{L}$ be a first order language. An interpretation for $\mathbf{L}$ is a *Herbrand interpretation* if the following conditions are satisfied:

- (a) The universe of the interpretation is the Herbrand universe $U_{\mathbf{L}}$;
- (b) Constants in $\mathbf{L}$ are assigned to "themselves" in $U_{\mathbf{L}}$;
- (c) If f is an n -ary function in $\mathbf{L}$, then f is assigned to the mapping from $(U_{\mathbf{L}})^n$ into $U_{\mathbf{L}}$ defined by $(t_1, \dots, t_n) \mapsto f(t_1, \dots, t_n)$.

There is no restriction on assignments of predicates in $\mathbf{L}$, different Herbrand interpretations arise by taking different such assignments. When we define the truth value denoted by any formula of a language in Definition 2.12, there is a total ordering $\leq$ on the truthvalue set V_m of the alphabet because of the numerical development of L_m . Based on this ordering, we introduce the following partial ordering on the set of all Herbrand interpretations of a language $\mathbf{L}$. (The definitions of partial orders and total orders are given in the next chapter.)

Definition 2.21 Let $\mathbf{L}$ be a language with the truthvalue set $V_m = \{0, \frac{1}{m-1}, \dots, \frac{m-2}{m-1}, 1\}$ and $\mathcal{I}_{\mathbf{L}}$ be the set of all Herbrand interpretations of $\mathbf{L}$. Then the relation $\preceq$ on $\mathcal{I}_{\mathbf{L}}$ is defined as follows: for any $I, J \in \mathcal{I}_{\mathbf{L}}$, $I \preceq J$ iff $/a/I \leq /a/J$ for every ground atom $a \in B_{\mathbf{L}}$.

The above definition can be easily extended to interpretations with universes other than the Herbrand universe. Let I and J be two Herbrand interpretations for a language $\mathbf{L}$. We say I is smaller than J if $I \preceq J$.

Definition 2.22 Let $\mathbf{L}$ be a language and S a set of sentences of $\mathbf{L}$. A *Herbrand model* for S is a Herbrand interpretation for $\mathbf{L}$ which is a model for S .

Definition 2.23 Let S be a set of sentences of a language $\mathbf{L}$ and M be a Herbrand model for S .

- (a) M is said to be a minimal Herbrand model if there is no smaller Herbrand model, i.e. for any Herbrand model N for S , $N \preceq M$ implies that $/a/N = /a/M$ for every $a \in B_L$.
- (b) M is said to be the least Herbrand model if for any Herbrand model N for S , $M \preceq N$.

Clearly, when the least Herbrand model for S exists, it is the unique minimal Herbrand model for S .

Definition 2.24 A formula is *open* if it contains no quantifiers. A formula is in *prenex form* if it has the form $Qx_1, \dots, Qx_n F$, where Qx_i is either $\forall x_i$ or $\exists x_i$; $x_1, \dots, x_n$ are distinct and F is open. A formula is in *universal prenex form* if it is in prenex form and contains no existential quantifiers.

Next we show that in order to prove unsatisfiability of a set of sentences in universal prenex form, it suffices to consider only Herbrand interpretations.

Proposition 2.2 *Let S be a set of sentences of a language L in universal prenex form and suppose S has a model M . Then S has a Herbrand model M' such that $/a/M = /a/M'$ for every $a \in B_L$.*

Proof: Let I be an interpretation of L . We define a Herbrand interpretation I' of L as follows. Each n -ary predicate p of L is assigned to the mapping from $(U_L)^n$ into the truthvalue set V of L defined by $(t_1, \dots, t_n) \mapsto /p(t_1, \dots, t_n)/_I$. Thus we have

$$/p(t_1, \dots, t_n)/_{I'} = /p(t_1, \dots, t_n)/_I, \text{ where } \{t_1, \dots, t_n\} \subseteq U_L.$$

Then it is straightforward to show by structural induction on the number of connectives that if I is a model for S , so is I' . $\square$

Proposition 2.3 *Let S be a set of sentences of a language L in universal prenex form. Then S is unsatisfiable iff S has no Herbrand models.*

Proof: If S has a model, then it has a Herbrand model by Proposition 2.2. $\square$

It should be noted that neither Proposition 2.2 nor Proposition 2.3 holds if S were not restricted to sentences in universal prenex form. In other words, if S is a set of arbitrary sentences, it is not generally possible to show the unsatisfiability of S by restricting attention to Herbrand interpretations. For S restricted in universal prenex form, Herbrand interpretations suffice.

Example 2.3 Let S be $\{p(a), \exists x \neg p(x)\}$. Note that the second sentence in S is not in universal prenex form. Upon any universe U with no fewer than two elements, we can construct a model for S . It suffices to let U be the set $\{1, 2, \dots\}$, assign a to 1, and assign p the mapping which maps 1 to **t** and 2 to **f**. Clearly this gives a model for S .

However, any interpretation I such that the universe of I has only one element is not a model for S . If a denotes the element and the mapping assigned to p maps the element to **t**, then I is not a model for $\exists x \neg p(x)$. Therefore, when a is the only function symbol of the language $\mathbf{L}$, S does not have a Herbrand model because $B_{\mathbf{L}} = \{a\}$ has only one element.

We now introduce the concept of Herbrand instantiation.

Definition 2.25 A *substitution* θ is a finite set of the form $\{x_1/t_1, \dots, x_n/t_n\}$, where each x_i is a variable, each t_i is a term distinct from x_i and the variables $x_1, \dots, x_n$ are distinct. The substitution may also be denoted by $(\vec{x}/\vec{t})$, where $\vec{x} = (x_1, \dots, x_n)$ and $\vec{t} = (t_1, \dots, t_n)$.

Definition 2.26 Let $\theta = \{x_1/t_1, \dots, x_n/t_n\}$ be a substitution and F be a term or open formula. Then $F\theta$, the *instance* of F by θ is the expression obtained from F by simultaneously replacing each occurrence of the variable x_i in F by the term t_i ($i = 1, \dots, n$). If $F\theta$ is ground, then $F\theta$ is called a *ground instance* of F .

Definition 2.27 Let $G : \forall y_1, \dots, \forall y_m F$ be a formula of a language $\mathbf{L}$ in universal prenex form, where F is an open formula. Let $\vec{x} = (x_1, \dots, x_n)$ be all the distinct variables appearing in F , and $\vec{t} = (t_1, \dots, t_n) \in (U_{\mathbf{L}})^n$ be a n -tuple of ground terms in $\mathbf{L}$. Then $F(\vec{x}/\vec{t})$ is called an *instantiated formula* of G by $\vec{t}$. Let $[G]_{\mathbf{L}}$ denote the set of all instantiated formulae of G . Then

$$[G]_{\mathbf{L}} = \{F(\vec{x}/\vec{t}) : \vec{t} \in (U_{\mathbf{L}})^n\}$$

We say a formula is ground if it contains no variable. Clearly, an instantiated formula is ground since every occurrence of a variable is replaced with a ground term. When the Herbrand universe is infinite, the set of all instantiated formula of a formula in universal prenex form is an infinite set of ground formulae.

The concept of instantiated formulae of a formula in prenex form can be easily extended to a set of formulae in prenex form.

Definition 2.28 Let S be a set of formulae of a language $\mathbf{L}$ in universal prenex form. Then the Herbrand instantiation of S is defined as the set $[S]_{\mathbf{L}}$ of all instantiated formulae of S

$$[S]_{\mathbf{L}} = \bigcup_{G \in S} [G]_{\mathbf{L}}$$

Proposition 2.4 *Let S be a set of sentences of a language $\mathbf{L}$ in universal prenex form and I be a Herbrand interpretation of $\mathbf{L}$. I is a model for S iff I is a model for $[S]_{\mathbf{L}}$.*

Proof: Let $G \in S$ be $\forall x_1, \dots, \forall x_n F$, where F is an open formula, and $\vec{x} = (x_1, \dots, x_n)$ are all the distinct variables appearing in F . We prove by induction on the number n that $/G/I = \min\{/g/I : g \in [G]_{\mathbf{L}}\}$.

When $n = 0$, G itself is the only sentence in $[G]_{\mathbf{L}}$. Therefore, the result is true.

Suppose when $n = k$, the result is true. When $n = k + 1$, let G/t stand for $\forall x_2, \dots, \forall x_{k+1} F(x_1/t)$ for an term $t \in U_{\mathbf{L}}$. Then

$$\begin{aligned} /G/I &= \min\{/(G/t)/I : t \in U_{\mathbf{L}}\} \text{ by definition of } \forall \\ &= \min\{\min\{/g/I : g \in [G/t]_{\mathbf{L}}\} : t \in U_{\mathbf{L}}\} \text{ by inductive assumption} \\ &= \min\{/g/I : g \in [G/t]_{\mathbf{L}}, t \in U_{\mathbf{L}}\} \\ &= \min\{/g/I(g) : g \in [G]_{\mathbf{L}}\} \end{aligned}$$

Therefore, $/G/I = \min\{/g/I : g \in [G]_{\mathbf{L}}\}$ for any n . Thus I is a model for G iff I is a model for $[G]_{\mathbf{L}}$. Furthermore, I is a model for S iff I is a model for $[S]_{\mathbf{L}}$. $\square$

The above proposition shows that S and $[S]_{\mathbf{L}}$ are equivalent when restricted to Herbrand interpretations. Lifting the restriction, the equivalence does not hold. However, we have the following two propositions.

Proposition 2.5 *Let S be a set of sentences of a language $\mathbf{L}$ in universal prenex form. Then $S \models_{\mathcal{L}} [S]_{\mathbf{L}}$.*

Proof: Let $G \in S$ be $\forall x_1, \dots, \forall x_n F$, where F is an open formula, and $\vec{x} = (x_1, \dots, x_n)$ are all the distinct variables appearing in F , and I be any interpretation of $\mathbf{L}$ with universe U . We prove by induction on the number n that $/G/I \leq \min\{/g/I : g \in [G]_{\mathbf{L}}\}$.

When $n = 0$, G itself is the only sentence in $[G]_{\mathbf{L}}$. Therefore, the result is true.

Suppose when $n = k$, the result is true. When $n = k + 1$, let G/i stand for $\forall x_2, \dots, \forall x_{k+1} F(x_1/i)$ for an item $i \in U$. Then

$$\begin{aligned} /G/I &= \min\{/(G/i)/I : i \in U\} \text{ by definition of } \forall \\ &\leq \min\{\min\{/g/I : g \in [G/i]_{\mathbf{L}}\} : i \in U\} \text{ by inductive assumption} \\ &= \min\{/g/I : g \in [G/i]_{\mathbf{L}}, i \in U\} \\ &\leq \min\{/g/I : g \in [G]_{\mathbf{L}}\} \end{aligned}$$

Therefore, $/G/I \leq \min\{/g/I : g \in [G]_{\mathbf{L}}\}$ for any n . Thus If I is a model for G , i.e. $/G/I = \mathbf{t}$, then $\min\{/g/I : g \in [G]_{\mathbf{L}}\} = \mathbf{i}$. Hence, I is a model for $[G]_{\mathbf{L}}$. Furthermore, If I is a model for G , then I is a model for $[S]_{\mathbf{L}}$. $\square$

The reverse of Proposition 2.5 does not hold as indicated by the next example.

Example 2.4 Suppose a is the only function of a language. Let I be an interpretation with the universe $\{i_1, i_2\}$ such that $a/I = i_1$ and $p(i_1) = \mathbf{t}$ and $p(i_2) = \mathbf{f}$. Let G be $\forall x p(x)$. Then I is a model for $[G]_{\mathbf{L}}$ but it is not a model for G .

Proposition 2.6 Let S be a set of sentences of a language $\mathbf{L}$ in universal prenex form and g be a ground formula. Then $S \models_{\mathcal{L}} g$ implies $[S]_{\mathbf{L}} \models_{\mathcal{L}} g$.

Proof: Suppose $S \models_{\mathcal{L}} g$. Let M be a model for $[S]_{\mathbf{L}}$. By Proposition 2.2, there is a Herbrand model M' for $[S]_{\mathbf{L}}$ such that $a/M = a/M'$ for every $a \in B_{\mathbf{L}}$. By Proposition 2.4, M' is a model for S . Since $S \models_{\mathcal{L}} g$, M' is a model for g . Because $a/M = a/M'$ for every $a \in B_{\mathbf{L}}$, M is a model for g . Hence, $[S]_{\mathbf{L}} \models_{\mathcal{L}} g$. $\square$

Proposition 2.5 and Proposition 2.6 indicate that the Herbrand instantiation $[S]_{\mathbf{L}}$ of S is a ground representation of S .

2.3 Logic Programs

2.3.1 Syntax of Logic Programs

First of all, we introduce the syntax of logic programming.

Definition 2.29 A general program clause α of a language $\mathbf{L}$ is defined as a formula of the form

$$\forall x_1 \dots \forall x_k (K \leftarrow L_1 \wedge \dots \wedge L_m \wedge \mathbf{not} L_{m+1} \wedge \dots \wedge \mathbf{not} L_{m+n})$$

where K is a formula containing no $\forall$ and $\exists$, and each L_i is a literal of $\mathbf{L}$ and $x_1, \dots, x_k$ are all the variables occurring in K and L_i . K is called the *head*, $L_1 \wedge \dots \wedge L_m \wedge \mathbf{not} L_{m+1} \wedge \dots \wedge \mathbf{not} L_{m+n}$ is called the *body*, and each L_i , $i \geq 1$, is called a premise of α . A *general logic program* is defined as a finite set of general program clauses. A general logic program is said to be

- (a) *extended* if the head and all premises of each of its clauses are literals.
- (b) *normal* if the head and all premises of each of its clauses are atoms.
- (c) *definite program* if it is normal and $n = 0$ (with no **not**).

The formulas in an extended (normal and definite) program are called extended (normal and definite) program clauses correspondingly.

When there are no needs to distinguish extended, normal and definite programs, we simply say programs.

Rigorously speaking, an extended program clause is not a sentence of the language $\mathbf{L}$ since we introduce a new symbol **not**. Traditionally, the intuitive meaning of **not** l is “negation of l as failure” or “no evidences that l is true”. It is one of the major objectives and contributions of this dissertation to define the semantics of **not**. Roughly speaking, **not** l means that l can be justifiably assumed false in the semantics introduced later in this dissertation.

Because extended program clauses are the form of formulae used in logic programming, we conveniently adopt a special notation for program clauses. Throughout, we denote the extended program clause

$$\forall x_1 \dots \forall x_k (K \leftarrow L_1 \wedge \dots \wedge L_m \wedge \mathbf{not} L_{m+1} \wedge \dots \wedge \mathbf{not} L_{m+n})$$

in the above definition by

$$K \leftarrow L_1, \dots, L_m, \mathbf{not} L_{m+1}, \dots, \mathbf{not} L_{m+n}$$

The informal meaning of $\alpha: K \leftarrow L_1, \dots, L_m, \mathbf{not} L_{m+1}, \dots, \mathbf{not} L_{m+n}$ is that “for any assignment of individuals in the universe to variables in α , if $L_1, \dots, L_m$ are true and $L_{m+1}, \dots, L_{m+n}$ are not ‘true’ (under the semantics of logic programming), then K is true”. The clause α is considered a definition of the head K . On the other hand, a unit clause $K \leftarrow$, which is a special case of a program clause, means that “ K is true for any assignment of individuals in the universe to variables in K ”.

Clearly, the concept of an instantiated formula of a sentence defined in Definition 2.27 can be naturally extended to an extended program clause. Now we proceed to define the Herbrand instantiation of a program, which is a ground representation of a program.

Definition 2.30 Let Π be a program of a language $\mathbf{L}$. The *Herbrand instantiation* $[\Pi]_{\mathbf{L}}$ of Π is defined as the set of all (ground) program clauses that are instantiated formulae of Π . I.e., it is the set

$$[\Pi]_{\mathbf{L}} = \bigcup_{\alpha \in \Pi} [\alpha]_{\mathbf{L}}$$

A program clause in the Herbrand instantiation of Π is called an instantiated clause of Π .

2.3.2 Partial Model Semantics

We now discuss the semantics of logic programming. Let Π' be the set of sentences obtained from an extended logic program Π by replacing each **not** with $\neg$. ($\neg\neg a$ is

considered a.) Π' is called the corresponding logical formula of Π . Without causing confusion, we also refer to Π' as an extended logic program and sentences in Π' as program clauses, and use Π to denote its corresponding logical formula.

Definition 2.31 Let Π be a program of a language L . Π is said to be consistent in a logic $\mathcal{L}$ if its corresponding logical formula Π' is consistent in a logic $\mathcal{L}$.

Clearly, we are only interested in consistent programs. In the study of declarative semantics for logic programming, one major approach is to identify some model, called canonical model, or a class of models, which may be called the intended models, as the declarative semantics of a program. These models are usually Herbrand models.

The semantics of logic programming is to determine from a logic program what should be inferred true, what should be inferred false, and what should be inferred neither true nor false (because of incomplete information). A partial interpretation (defined below) describes a world in which some ground atoms are true, some are false, and others are neither true nor false.

Definition 2.32 Let L be a first order language and B_L the Herbrand base of L . A *partial interpretation* I of L is defined as a pair $\langle T, F \rangle$, where both T and F are subsets of B_L , i.e. $T \subseteq B_L$ and $F \subseteq B_L$. A *total interpretation* is defined as a partial interpretation $\langle T, F \rangle$ such that T and F partition B_L . A partial interpretation $\langle T, F \rangle$ is *consistent* if T and F are disjoint, i.e. $T \cap F = \emptyset$.

For a partial interpretation $I = \langle T, F \rangle$, T is intended to contain atoms true in I , and F is intended to contain atoms false in I while the truth value of the remaining atoms is *undefined*. With such an intended meaning for a partial interpretation, there is a natural one-to-one correspondence between partial interpretations and Herbrand interpretations of a 3-valued language. Similarly, there is a one-to-one correspondence between total interpretations and 2-valued Herbrand interpretation of any first order languages (not necessarily with only two truth values). A 2-valued first order language L is just a special case.

Definition 2.33 Two *natural orderings* $\preceq$ and $\subseteq$ on the set of partial interpretations are defined as follows: for any partial interpretations $I_1 = \langle T_1, F_1 \rangle$ and $I_2 = \langle T_2, F_2 \rangle$,

$$I_1 \preceq I_2 \text{ iff } T_1 \subseteq T_2 \text{ and } F_1 \supseteq F_2.$$

$$I_1 \subseteq I_2 \text{ iff } T_1 \subseteq T_2 \text{ and } F_1 \subseteq F_2.$$

i.e. $I_1 \preceq I_2$ means that T_1 is a subset of T_2 and F_1 is a superset of F_2 , and $I_1 \subseteq I_2$ means that T_1 and F_1 are subsets of T_2 and F_2 respectively. $\preceq$ is called the *standard ordering* while $\subseteq$ is called *F-ordering* (F for Fitting). When $I_1 \subseteq I_2$, we say that I_2 is an extension of I_1 .

We stated that each consistent partial interpretation identifies a Herbrand interpretation of a 3-valued language, and vice versa. With this identification, the standard ordering on consistent partial interpretations is equivalent to the ordering $\preceq$ defined in Definition 2.21 on Herbrand interpretations. Let J_1 and J_2 be Herbrand interpretations of a 3-valued language and I_1 and I_2 be the corresponding partial interpretations. Then $J_1 \preceq J_2$ iff $I_1 \preceq I_2$. This is the very reason that $\preceq$ is called the standard ordering. Although the symbols $\subseteq$ and $\preceq$ have been used for more than one purpose, any confusion can be avoided from the context.

Definition 2.34 Let Π be a program of a first order language $\mathbf{L}$ and M be a partial interpretation of $\mathbf{L}$. M is said to be a *partial model* of Π if there exists a total interpretation I of $\mathbf{L}$ such that I is an extension of M , i.e. $M \subseteq I$ and the corresponding Herbrand interpretation of I is a model of Π .

Now we proceed to define the partial model semantics for logic programming. The approach is model-theoretical. A model-theoretical approach to the theory of deductive database is based on the idea of representing a database as a set Π of sentences of first order logic and associating with Π a set of one or more models of Π which describe the declarative meaning of the database, i.e. its semantics. A sentence is considered true in the database iff it is satisfied in all the models in the set [69]. In our context, Π is a logic program and the models being considered are partial models of Π . Naturally, we have the following definition.

Definition 2.35 Let $\mathbf{L}$ be a first order language and let $\mathcal{I}_{\mathbf{L}}$ denote the set of all partial interpretations of $\mathbf{L}$ and $\mathcal{C}$ denote a class of programs of $\mathbf{L}$. By a *declarative semantics for a class $\mathcal{C}$* of programs, we mean a mapping $SEM : \mathcal{C} \mapsto 2^{\mathcal{I}_{\mathbf{L}}}$ from the set $\mathcal{C}$ of concerned programs to the power set $2^{\mathcal{I}_{\mathbf{L}}}$ of the set of partial interpretations. Moreover, SEM is called a *partial model semantics* if it satisfies the condition that for any program $\Pi \in \mathcal{C}$, $I \in SEM(\Pi)$ implies that I is a partial model of Π .

The class $\mathcal{C}$ is not always the set of all programs. It is a desirable property of semantics for logic programming that the function SEM is defined on the set of all programs. For a declarative semantics SEM and a program Π , each partial interpretation in $SEM(\Pi)$ is considered an intended interpretation of the program Π , and represent a possible world described by the program Π . It turns out that most semantics proposed are partial model semantics. For queries against a knowledge base represented by a program, only those answers which are correct with respect to every partial interpretation in $SEM(\Pi)$ are considered correct. Only knowledge represented by all the partial interpretations in $SEM(\Pi)$ is to be inferred from Π .

2.3.3 Queries and Answers

Definition 2.36 A *query* is an open conjunctive formula of the form $L_1 \wedge \dots \wedge L_n$, where each L_i is a literal. A query may also be denoted by $L_1, \dots, L_n$.

Definition 2.37 Let $\mathbf{L}$ be a first order language and $I = \langle T, F \rangle$ a partial interpretation of $\mathbf{L}$. We introduce the following notation.

- (a) For any ground atom $a \in B_{\mathbf{L}}$, we write $I \models a$ iff $a \in T$, and $I \models \neg a$ iff $a \in F$;
- (b) For any ground query $Q : l_1, \dots, l_n$, we write $I \models Q$ iff $I \models l_i$ for each i ;
- (c) For any query Q , we write $I \models \forall Q$ iff for every instantiated formula $q \in [Q]_{\mathbf{L}}$, $I \models q$;

The above notations may easily be extended to a set of partial interpretations.

Definition 2.38 Let $\mathbf{L}$ be a first order language and S a set of partial interpretations of $\mathbf{L}$.

- (a) For a ground query Q , we write $S \models Q$ iff for every $I \in S$, $I \models Q$;
- (b) For a query Q , we write $S \models \forall Q$ iff for every $I \in S$, $I \models \forall Q$;

Definition 2.39 Let Q be a query. An *answer substitution* for Q is a substitution for variables of Q .

Definition 2.40 Let SEM be a semantics for a class $\mathcal{C}$ of programs, Π be a program in $\mathcal{C}$ and Q be a query. An answer substitution θ for Q is said to be a *correct answer substitution* for Q wrt $SEM(\Pi)$ if $SEM(\Pi) \models \forall(Q\theta)$.

By Definition 2.37 and 2.38, θ is a correct answer substitution for Q wrt $SEM(\Pi)$ iff for every partial interpretation I in $SEM(\Pi)$ and every element q in $[Q\theta]_{\mathbf{L}}$, $I \models q$, i.e. q is true in I . This definition of correct answer substitution captures the intuitive meaning of a "correct answer" since $SEM(\Pi)$ is taken as the set of possible worlds described by Π and all elements in $[Q\theta]_{\mathbf{L}}$ are knowledge represented by $SEM(\Pi)$. It provides a declarative understanding of the desired output from a program and a query.

Correctness of answer substitutions is defined wrt a declarative semantics for logic programming. Thus an answer substitution correct wrt a semantics may not be correct wrt another semantics. Since correct answer substitutions provide a declarative understanding of a program and a query wrt a declarative semantics, they also provide a criterion for the equivalence of different semantics.

Definition 2.41 Let SEM_1 and SEM_2 be two semantics for a class $\mathcal{C}$ of programs. SEM_1 and SEM_2 are said to be equivalent if for any program $\Pi \in \mathcal{C}$ and any query Q , an answer substitution θ for Q is correct wrt $SEM_1(\Pi)$ iff θ is correct wrt $SEM_2(\Pi)$.

We introduce some setwise operations on partial interpretations. Let $I_1 = \langle T_1, F_1 \rangle$ and $I_2 = \langle T_2, F_2 \rangle$. The intersection and union of I_1 and I_2 are defined as follows:

$$I_1 \cap I_2 = \langle T_1 \cap T_2, F_1 \cap F_2 \rangle;$$

$$I_1 \cup I_2 = \langle T_1 \cup T_2, F_1 \cup F_2 \rangle.$$

Proposition 2.7 Let SEM be a semantics for a class $\mathcal{C}$ of programs. Let the semantics SEM' be defined as the intersection of all the partial interpretations in SEM . I.e. for a program $\Pi \in \mathcal{C}$,

$$SEM'(\Pi) = \bigcap_{I \in SEM(\Pi)} I, \text{ for every } \Pi \in \mathcal{C}.$$

Then SEM and SEM' are equivalent.

Proof: Let $Q : L_1, \dots, L_n$ be a query and θ be a answer substitution for Q .

Q is correct wrt $SEM(\Pi)$
iff $SEM(\Pi) \models \forall(Q\theta)$
iff $I \models \forall(Q\theta)$, for every $I \in SEM(\Pi)$
iff $I \models q$, for every $q \in [Q\theta]_{\mathbf{L}}$ and every $I \in SEM(\Pi)$
iff $\cap_{I \in SEM(\Pi)} I \models q$, for every $q \in [Q\theta]_{\mathbf{L}}$
iff $\cap_{I \in SEM(\Pi)} I \models \forall(Q\theta)$
iff $SEM'(\Pi) \models \forall(Q\theta)$
iff Q is correct wrt $SEM'(\Pi)$.

□

By Proposition 2.7, it would not change the semantic meaning of a program just to consider the intersection of the partial interpretations representing the semantics. The intersection is one single partial interpretation, and sometimes it is convenient to consider the intersection. This proposition also shows that in the context of logic programming, we adopt the notion of skeptical inference or reasoning. In other words, an answer is considered correct wrt a semantics iff it is considered correct in *every* intended partial interpretation of a program.

Chapter 3

A Framework for Semantics of Logic Programming Based on Hypothetical Reasoning

In this chapter, we shall propose a framework for semantics of logic programming based on hypothetical reasoning. The idea is to postulate criteria and conditions to justify acceptable hypotheses, and then to define semantics of logic programs in terms of acceptable hypotheses. In the framework, hypotheses are of the form of a tuple $\langle P, N \rangle$, where P is a set of positive hypotheses, and N is a set of negative hypotheses. The acceptability of a hypothesis tuple is justified by its *admissibility*, which is a concept introduced in this chapter, and the semantics of a logic program is defined by a set of admissible hypothesis tuples, saturated with all acceptable hypotheses. The semantics of the underlying logic is reflected in defining the admissibility of hypothesis tuples. It turns out that it plays an important role in semantics defined for logic programming.

Each saturated hypothesis tuple corresponds to a possible world described by a logic program. For different viewpoints of knowledge representation, different possible worlds may be considered the intended worlds described by the program. When a skeptical, credulous, or omniscient view is adopted, the smallest saturated tuple, the maximal saturated tuples, and the stable tuples, are used to define the semantics respectively.

The framework can be applied to disjunctive programs on the basis of a new concept of *semantical instance*. It can also be applied to extended programs by using the *oriented consequences* of a program. Therefore, the framework is defined for the most general form of logic programs, extended disjunctive programs.

Based on the framework, we shall show that most existing semantics fit into the

framework, and also propose new semantics.

3.1 A Simple Framework for Hypothetical Reasoning

Poole proposed a framework of a simple case of hypothetical reasoning, namely where a user provides the form of possible hypotheses the user is prepared to accept in an explanation [64]. The intuitive idea is that given a representation system, which is a set of facts known to be true, and a set of possible hypotheses a user is willing to accept as a part of an explanation to predict expected observations, the user reasons by identifying a restricted set of possible hypotheses in the set to form an explanation. Forming such an *explanation* should be viewed as forming a “scientific theory” from the restricted set of possible hypotheses. Furthermore, it is useful to view the explanation as a scenario in which some goal is true. Since not all sets of possible hypotheses may be used to form scenarios, the user provides restrictions on what is acceptable in such scenarios. A simple example of restrictions is that the user would not want to find an explanation for observations which are not consistent with the representation system (i.e. would not predict anything known to be false). In other words, the explanation, or the “scientific theory” should be consistent.

This section briefly presents the simple framework of hypothetical reasoning proposed by Poole in [64]. The framework is presented in terms of the classical first order logic $\mathcal{CL}$, although the idea is independent of the particular logic used.

It is assumed that the sets $\mathcal{F}$ and Δ are given. $\mathcal{F}$ is the representation system consisting of a set of sentences, or closed formula, which are considered as “facts” or “knowledge” known. It is also assumed that the set $\mathcal{F}$ is consistent, or satisfiable. Δ is a set of formulae, called the set of *possible hypotheses*. Any ground instance of these formulae can be used as a hypothesis if it is consistent. A set of ground instances of these formulae is also called a hypothesis set. We say that $(\mathcal{F}, \Delta)$ is a hypothetical reasoning system, or shortly, a reasoning system.

Definition 3.1 [64, Definition 2.1] A *scenario* of $(\mathcal{F}, \Delta)$ is a set $H \cup \mathcal{F}$ where H is a set of ground instances of elements of Δ such that $H \cup \mathcal{F}$ is consistent.

Definition 3.2 [64, Definition 2.2] If S is a sentence, or closed formula, then an *explanation* of S from $(\mathcal{F}, \Delta)$ is a scenario of $(\mathcal{F}, \Delta)$ which implies S .

Definition 3.3 [64, Definition 2.3] An *extension* of $(\mathcal{F}, \Delta)$ is the set of logical consequences of a maximal (wrt set inclusion) scenario of $(\mathcal{F}, \Delta)$.

Extensions are "treated" as the semantics of the hypothetical reasoning system $(\mathcal{F}, \Delta)$. We now review some properties of the above definitions. Proposition 3.1 follows directly from the compactness theorem of the classical first order logic.

Proposition 3.1 [64, Lemma 2.4] *If S is explicable from $(\mathcal{F}, \Delta)$, then it is explicable using a finite scenario.*

Proposition 3.2 (Monotonicity) [64, Lemma 2.5] *If $H_1 \subseteq H_2$ and $H_1 \cup \mathcal{F} \models_{\mathcal{CL}} S$, then $H_2 \cup \mathcal{F} \models_{\mathcal{CL}} S$.*

The equivalence between explicability and being in an extension is established by the following proposition.

Proposition 3.3 [64, Theorem 2.6] *S is explicable if and only if S is in some extension.*

A logic program can be viewed as a representation and reasoning system. There are arguments saying that a reasoning system must do some hypothesizing and testing. There has been much study of this, particularly in the area of scientific theory formation.

In [64], it is shown that theory formation from a fixed set of hypotheses is a natural and simple characterization of default reasoning. We show that theory formation is also a natural and simple approach to the semantics for logic programming. We will propose a very general framework for semantics of extended logic program based on theory formation from the representation system — a logic program and a restricted set of possible hypotheses.

3.2 A Framework for Semantics of Logic Programming

In this section, we show how Poole's framework is applied to logic programming, and explain why some nontrivial adjustments have to be made in order for the framework to make sense in the context of logic programming. Since the idea of the framework is independent of the logic used, we do not restrict ourselves to the classical first order logic. We use $\mathcal{L}$ to denote the logic used. So most definitions and concepts are introduced using $\mathcal{L}$ to denote the underlying logic. Lukasiewiczian many-valued first order logic L_m for any $m \geq 2$ is used to present examples. Other logics will be considered in later chapters.

In Poole's framework, one of the most difficult questions of science, namely "How is the set of all possible hypotheses generated?", is deliberately avoided. We try to provide an answer to the question in the context of logic programming. An extended program of a language $\mathbf{L}$ is a finite set of formulas of the form

$$L_0 \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_{m+n},$$

where the L_i 's are literals. We intend to define the semantics in terms of acceptable hypotheses, i.e., in terms of literals that are justifiably assumed TRUE and literals that are justifiably assumed FALSE. Thus a natural choice of possible hypotheses is the set of all literals. When the extended program is normal, i.e., when all the L_i 's are atoms, its semantics is defined in terms of atoms that are justifiably assumed TRUE and atoms that are justifiably assumed FALSE. Thus, a natural choice of possible hypotheses is the set of all atoms. Let δ_Π denote the set of all possible hypotheses. Then δ_Π is the set of literals of $\mathbf{L}$ when $\neg$ occurs in Π , and δ_Π is the set of all atoms of $\mathbf{L}$ when Π is normal. Since only ground instances of hypotheses are used, it would simplify our presentation to consider the set of all ground literals or atoms. Let $B_{\mathbf{L}}$ be the Herbrand base of $\mathbf{L}$ and $Lit_{\mathbf{L}}$ stand for the set of all ground literals of $\mathbf{L}$, i.e. $Lit_{\mathbf{L}} = B_{\mathbf{L}} \cup \{\neg a : a \in B_{\mathbf{L}}\}$. Then $\delta_\Pi = Lit_{\mathbf{L}}$ when $\neg$ occurs in Π , and $\delta_\Pi = B_{\mathbf{L}}$ when Π is normal.

We now introduce a hypothetical reasoning system for logic programming. A hypothesis tuple of Π is defined as a tuple $\langle P, N \rangle$ such that $P, N \subseteq \delta_\Pi$. The set P is the set of possible hypotheses assumed true while N is the set of possible hypotheses assumed false. Let Δ_Π denote $\langle \delta_\Pi, \delta_\Pi \rangle$. I.e., any $H \subseteq \Delta_\Pi$ is a tuple $\langle P, N \rangle$ such that $P, N \subseteq \delta_\Pi$. Then, the reasoning system we are considering is of the form (Π, Δ_Π) , where Π , corresponding to $\mathcal{F}$, is a program in a language $\mathbf{L}$ and Δ_Π , corresponding to the set Δ of possible hypotheses, is the tuple of all hypothesis set of Π . Clearly, in the reasoning system (Π, Δ_Π) , hypotheses are represented as hypothesis tuples instead of hypothesis sets.

We now introduce some notations. $\neg l$ is assumed to denote a if l is of the form $\neg a$ where $a \in B_{\mathbf{L}}$ is an atom. Let $H = \langle P, N \rangle \subseteq \Delta_\Pi$. Then H_P and H_N stands for $\langle P, \emptyset \rangle$ and $\langle \emptyset, N \rangle$ respectively. In other words, H_P corresponds to the set of hypotheses assumed positively while H_N corresponds to the set of hypotheses assumed negatively. For a literal l , we write $l \in H$ if $l \in P$ or $\neg l \in N$. In other words, $l \in H$ means that l is assumed by H . Without causing confusion, we also use H to denote the set $\{l : l \in H\}$, which is actually $P \cup \{\neg l : l \in N\}$.

3.2.1 Incorporating Hypotheses

First of all, we consider how to incorporate hypotheses with the representation system.

Let us consider the following example.

Example 3.1 Let Π be $\{a \leftarrow \text{not} b\}$, $H_1 = \langle \emptyset, \{a\} \rangle$ and $H_2 = \langle \emptyset, \{b\} \rangle$. In Poole's framework, set union is used to incorporate hypotheses. Then $H_1 \cup \Pi = \{\neg a\} \cup \{a \leftarrow \neg b\}$ is an explanation of b since $H_1 \cup \Pi \models_{L_m} b$ and $H_1 \cup \Pi$ is consistent. Similarly, $H_2 \cup \Pi$ is an explanation of a . However, in the context of logic programming, Π should not be considered a proper mechanism to infer b from $\neg a$. Thus H_1 should not be considered a proper hypothesis to predict b . Nevertheless, H_2 predicts a properly from Π .

Example 3.1 shows that the set union is not a proper incorporation of the representation system and hypotheses in the context of logic programming. Let $\uplus$ denote the incorporating operator for a reasoning system. In the general framework introduced in the last section, $\uplus$ is simply the set union $\cup$. We now introduce the incorporating operator $\uplus$ suitable for logic programming.

Definition 3.4 Let (Π, Δ_Π) be a hypothetical reasoning system, where Π is an extended logic program. Then for any hypothesis tuple $H = \langle P, N \rangle \subseteq \Delta_\Pi$, $H \uplus \Pi$ is defined as the set obtained from $[\Pi]_L$ by deleting each **not** l from the clauses in $[\Pi]_L$ if $l \in N$.

In the above definition, only the negative hypotheses in N are incorporated with the program. The positive hypotheses in P will be used when we extend the above definition to disjunctive programs.

Recall that the logical formula of a program Π is obtained by replacing **not** with $\neg$. The concept of logical formula can easily be extended to a set S of sentences that contain **not**. Without causing confusion, we also use S to denote its logical formula. By Definition 2.31, a program consistent in $\mathcal{L}$ is a program whose logical formula is consistent in $\mathcal{L}$. We are only interested in consistent programs. It is assumed that we consider consistent programs only, unless otherwise stated.

Example 3.2 Consider Example 3.1 again. $H_1 \uplus \Pi$ is $\{a \leftarrow \neg b\}$, i.e. Π itself. And $H_1 \uplus \Pi \not\models_{L_m} b$. On the other hand, $H_2 \uplus \Pi$ is $\{a\}$, and thus $H_2 \uplus \Pi \models_{L_m} a$. If H_1 and H_2 are considered as hypotheses and the incorporation of the program and the hypotheses are expressed by $H_1 \uplus \Pi$ and $H_2 \uplus \Pi$ respectively, then H_1 can not predict b while H_2 still predicts a . Thus H_1 is not to be accepted in an explanation for b while H_2 is for a .

Clearly, the above example indicates that the incorporating operator $\uplus$ reflects the rule-like behavior of $\leftarrow$.

Proposition 3.4 *Let Π be a program and $H \subseteq \Delta_\Pi$. Then for any S , $H \uplus \Pi \models_{L_m} S$ implies $H \cup \Pi \models_{L_m} S$.*

Proof: By Proposition 2.5, $\Pi \models_{L_m} [\Pi]_L$. Then $H \cup [\Pi]_L \models_{L_m} H \uplus \Pi$ follows from the fact that for any ground program clause $\alpha : l \leftarrow l_1, \dots, l_n$ and $\beta : l \leftarrow l_1, \dots, l_n, k_1, \dots, k_m$, $H \cup \{\beta\} \models_{L_m} \alpha$ if $\{k_1, \dots, k_m\} \subseteq H$. Therefore, $H \cup \Pi \models_{L_m} H \uplus \Pi$. $\square$

In general, we do not have $H \uplus \Pi \models_{L_m} H \cup \Pi$. However, we shall show in Proposition 3.37 that $H \uplus \Pi$ is as explanatory as $H \cup \Pi$ when Π is normal and H is L_m -saturated. (The concept of L_m -saturation will be introduced later in this chapter.) We shall also show that Definition 3.4 preserves most properties of the previous reasoning system. We show here the monotonicity.

Proposition 3.5 (Monotonicity) *If $H_1 \subseteq H_2 \subseteq \Delta_\Pi$ and $H_1 \uplus \Pi \models_{L_m} S$, then $H_2 \uplus \Pi \models_{L_m} S$.*

Proof: $H_2 \uplus \Pi \models_{L_m} H_1 \uplus \Pi$ follows from that fact that for any ground program clause $\alpha : l \leftarrow l_1, \dots, l_n$ and $\beta : l \leftarrow k_1, \dots, k_m$, $\alpha \models_{L_m} \beta$ when $\{l_1, \dots, l_n\} \subseteq \{k_1, \dots, k_m\}$. Then the result follows immediately. $\square$

In parallel with the simple framework in the last section, we could define the concepts of scenarios, explanations and extensions by replacing the set union $\cup$ with $\uplus$ in the corresponding definitions. However, concepts defined as such are not interesting. For the framework in the last section, consistency is the only restriction on hypotheses in a scenario. With the introduction of $\uplus$, such a restriction is lifted for any hypotheses of the reasoning system in terms of a logic program because for any hypothesis tuple H and any program Π , $H \uplus \Pi$ is always consistent for a normal program Π . It is obvious that not all hypothesis tuples should be accepted in explanations. Clearly, other restrictions must be introduced in order for the reasoning system to make proper sense. This is the task of the next few sections.

3.2.2 Noncontradictoriness

With the new incorporating operator $\uplus$ introduced, $H \uplus \Pi$ is consistent for any hypothesis tuple $H \subseteq \Delta_\Pi$ and any normal program Π . Clearly, we would not accept all hypotheses as reasonable explanations for expected observations. The next few sections introduce some restrictions on hypothesis tuples in order for the hypotheses to be accepted in explanations. On the other hand, these restrictions can also be viewed as justifications for the hypotheses to be accepted in explanations.

The consistency requirement for a scenario says that no hypotheses, that predict absurdity (i.e. things known to be false), should be accepted in explanations for any expected observations. If such hypotheses had been accepted in explanations, they would not only predict things known to be false but also predict the negation of the hypotheses themselves. Obviously, it is undesirable for hypotheses to predict the negation of themselves.

Definition 3.5 Let Π be a program and $H = \langle P, N \rangle \subseteq \Delta_\Pi$. H is said to be $\mathcal{L}$ -noncontradictory of Π , if $H \uplus \Pi \not\models_{\mathcal{L}} l$ for each $l \in N$.

It should be noted that $H \uplus \Pi \not\models_{\mathcal{L}} \neg l$ for each $l \in P$ is not a premise for H to be noncontradictory. The noncontradictoriness is a restriction on the negative hypothesis set N . Other restrictions will be introduced later for the positive hypothesis set P . Actually, we adopt the idea that the negative hypotheses should be maximized while the positive hypotheses should be minimized. Thus, P should contain only hypotheses it has to contain. This will be defined as positive foundedness later.

Proposition 3.6 Let $H_1 \subseteq H_2 \subseteq \Delta_\Pi$. If H_2 is L_m -noncontradictory, so is H_1 .

Proof: The result follows immediately from Proposition 3.5. $\square$

Proposition 3.7 Let $H \subseteq \Delta_\Pi$ be L_m -noncontradictory of a program Π . $H \uplus \Pi$ is consistent in the logic L_m .

Proof: The result follows immediately from Definition 3.5. $\square$

3.2.3 Positive Foundedness and False-Assumability

We argue that positive hypotheses should be minimized while negative hypotheses should be maximized. In common sense reasoning, a human being tends to accept negative hypotheses more easily and readily than positive hypotheses. An example of such reasoning is the closed world assumption adopted in database theory. Anything not specified in a database is readily accepted as a negative conclusion by a user of the database. This thesis mirrors the idea that "positive introspection should be minimized while negative introspection should be maximized." [97]. Positive foundedness is a property of a hypothesis tuple with minimized positive hypotheses while false-assumability provides a criterion to maximized negative hypotheses of the set.

Definition 3.6 Let Π be a program and $H = \langle P, N \rangle \subseteq \Delta_\Pi$. A literal $l \in \delta_\Pi$ is said to be *positively $\mathcal{L}$ -founded* of Π wrt H if $H \uplus \Pi \models_{\mathcal{L}} l$. H is also said to be *positively $\mathcal{L}$ -founded* of Π if every ground literal $l \in P$ is positively $\mathcal{L}$ -founded of Π wrt H .

Example 3.3 Consider the program $\Pi : \{a \leftarrow b, b \leftarrow a\}$ and hypotheses $H = \langle \{a, b\}, \emptyset \rangle$. Neither a nor b is positively L_m -founded since $H \uplus \Pi = \Pi$, $\Pi \not\models_{L_m} a$ and $\Pi \not\models_{L_m} b$. This indicates that neither a nor b are to be assumed true.

For a hypothesis tuple $\langle P, N \rangle$, Definition 3.4 and 3.6 indicate that N functions as a basis for positively founded literals in P since only literals in N have been incorporated. Example 3.3 shows that a hypothesis tuple, which is not positively founded, may predict unfounded positive conclusions. We should only be interested in positively founded hypotheses.

Given a hypothesis tuple $H = \langle P, N \rangle$, what else could be justifiably put in H ? I.e., what ground literals may be justifiably assumed true and what literals may be justifiably assumed false? Clearly, positive foundedness is a criterion for assuming literals to be true. Then how can a literal be justifiably assumed false? Dung proposed a simple but very effective idea [12]. Consider a ground literal l . There may exist a hypothesis tuple H_l such that $H_l \uplus \Pi \models_{\mathcal{L}} l$, where Π is a program. If it is the case that no such hypothesis tuple is compatible or consistent with H , we would say that $\neg l$ is quite acceptable as an additional hypothesis to H . In our framework, $\neg l$ is acceptable as a further hypothesis if l is not explicable from the reasoning system. This leads to the next definition.

Definition 3.7 Let Π be a program and $H \subseteq \Delta_\Pi$. A ground literal $l \in \delta_\Pi$ is said to be *$\mathcal{L}$ -false-assumable* of Π wrt H if for every $H_l \subseteq \Delta_\Pi$ satisfying $H_l \uplus \Pi \models_{\mathcal{L}} l$, there exists a literal $k \in H_l$ such that $H \uplus \Pi \models_{\mathcal{L}} \neg k$.

Example 3.4 c is L_m -false-assumable of the program $\Pi_4 \{c \leftarrow \text{nota}, \text{not}b, b \leftarrow \text{nota}, a \leftarrow \text{not}b\}$ wrt $\{\neg a\}$ or $\{\neg b\}$.

False-assumability wrt a hypothesis tuple H means unexplicability from a reasoning system by any hypothesis tuple which is compatible with H . Incompatibility is demonstrated by the fact that at least one hypothesis l assumed in such a tuple should not be accepted as a part of an explanation because the set H can predict $\neg l$.

The following Proposition 3.8 follows from the monotonicity of the reasoning system. It says that the stronger hypotheses preserve the false-assumability wrt weaker hypotheses.

Proposition 3.8 Let Π be a program and l be a ground literal. If $H_1 \subseteq H_2 \subseteq \Delta_\Pi$ and l is L_m -false-assumable of Π wrt H_1 , then l is L_m -false-assumable of Π wrt H_2 .

Proof: Because l is L_m -false-assumable of Π wrt H_1 , for any H_l satisfying $H_l \uplus \Pi \models_{L_m} l$, there exists $k \in H_l$ such that $H_1 \uplus \Pi \models_{L_m} \neg k$. By Proposition 3.5, $H_2 \uplus \Pi \models_{L_m} \neg k$. Hence, l is L_m -false-assumable of Π wrt H_2 . $\square$

False-assumability of a literal l wrt a hypothesis tuple H justifies the acceptance of the hypothesis $\neg l$ as a part of explanations. Positive foundedness defined by Definition 3.6 introduced a criterion to justify the acceptance of positive hypotheses. In order to justify the acceptance of H itself as a part of explanations, we introduce a criterion to justify the acceptance of negative hypotheses, based on the concept of false-assumability.

Definition 3.8 Let Π be a program and $H = \langle P, N \rangle \subseteq \Delta_\Pi$. H is said to be *negatively $\mathcal{L}$ -founded* of Π if every literal $l \in N$ is $\mathcal{L}$ -false-assumable of Π wrt H . H is said to be *$\mathcal{L}$ -founded* of Π if H is both positively and negatively $\mathcal{L}$ -founded.

All literals in N of a negatively $\mathcal{L}$ -founded H justify their false-assumability collectively on their own. Positive and negative foundedness justify the acceptance of positive and negative hypotheses respectively.

3.2.4 Admissibility

It seems that an $\mathcal{L}$ -founded set H of hypotheses can justify its acceptance as a part of explanations on itself. However, $\mathcal{L}$ -foundedness is not sufficient because an $\mathcal{L}$ -founded set H may not be $\mathcal{L}$ -noncontradictory. This is shown by the next example.

Example 3.5 $\langle \emptyset, \{p\} \rangle$ is L_m -founded of the program $\{p \leftarrow \text{not } p\}$, but it is not L_m -noncontradictory of Π .

The above discussion lead us to the following definition.

Definition 3.9 Let Π be a program and $H \subseteq \Delta_\Pi$. The hypothesis tuple H is said to be *$\mathcal{L}$ -admissible* of Π if H is both $\mathcal{L}$ -noncontradictory and $\mathcal{L}$ -founded of Π .

Lemma 3.9 Let $H \subseteq \Delta_\Pi$ be L_m -noncontradictory of a program Π . For $l \in \delta_\Pi$, $H \uplus \Pi \not\models_{L_m} l$ if l is L_m -false-assumable of Π wrt H .

Proof: Suppose l is L_m -false-assumable of Π wrt H and $H \uplus \Pi \models_{L_m} l$. Then $H_N \uplus \Pi \models_{L_m} l$ where $H_N = \langle \emptyset, N \rangle$. Thus there exists $k \in N$ such that $H \uplus \Pi \models_{L_m} \neg k$. This contradicts the fact that H is L_m -noncontradictory. $\square$

The following Proposition 3.10 shows the the relationship between foundedness and admissibility. It says that enlarging an admissible hypothesis tuple with founded literals would preserve the admissibility of the hypothesis tuple.

Proposition 3.10 Let $H = \langle P, N \rangle \subseteq \Delta_\Pi$ be L_m -admissible of a program Π , T be a set of positively L_m -founded literals of Π wrt H , and F be a set of L_m -false-assumable literals of Π wrt H . Then $H' = H \cup \langle T, F \rangle$ is L_m -admissible of Π .

Proof: Let $H' = \langle P', N' \rangle$. By Proposition 3.5, H' is positively L_m -founded. Since for each $l \in N'$, l is L_m -false-assumable wrt H , l is also L_m -false-assumable wrt H' by Proposition 3.8. Therefore, H' is negatively L_m -founded.

We then prove H' is L_m -noncontradictory. Suppose there exists $l \in N'$ such that $H' \cup \Pi \models_{L_m} \neg l$. Then $H'_N \cup \Pi \models_{L_m} \neg l$ by Definition 3.4, where $H'_N = \langle \emptyset, N' \rangle$.

Since $\neg l$ is L_m -false-assumable wrt H , there exists $k \in N'$ such that $H \cup \Pi \models_{L_m} k$. This contradicts Lemma 3.9 since k is L_m -false-assumable wrt H . Therefore, H' is L_m -noncontradictory. By Definition 3.8 and 3.9, H' is L_m -admissible. $\square$

3.2.5 $\mathcal{L}$ -Saturated Extension Semantics

We argued in Section 3.2.3 that positive hypotheses should be minimized while negative hypotheses should be maximized. Hence,

Definition 3.10 Let $H = \langle P, N \rangle \subseteq \Delta_\Pi$ be $\mathcal{L}$ -admissible of a program Π . H is said to be $\mathcal{L}$ -saturated of Π if for any literal $l \in \delta_\Pi$,

- (a) $l \in P$ iff l is positively $\mathcal{L}$ -founded wrt H ;
- (b) $l \in N$ iff l is $\mathcal{L}$ -false-assumable wrt H .

An $\mathcal{L}$ -admissible hypothesis tuple H is $\mathcal{L}$ -saturated if it contains every positively $\mathcal{L}$ -founded literal and every $\mathcal{L}$ -false-assumable literal wrt H . A saturated tuple is a hypothesis set that is maximized wrt the foundedness and preserves the admissibility. Now we define an $\mathcal{L}$ -preferred hypothesis tuple, which is maximized wrt F-ordering $\subseteq$ and preserves the admissibility.

Definition 3.11 Let $H \subseteq \Delta_\Pi$ be $\mathcal{L}$ -admissible of a program Π . H is said to be $\mathcal{L}$ -preferable of Π if for any $\mathcal{L}$ -admissible $H' \subseteq \Delta_\Pi$, $H \subseteq H'$ implies $H = H'$.

An $\mathcal{L}$ -preferable hypothesis tuple H is a maximal (wrt F-ordering) $\mathcal{L}$ -admissible hypothesis tuple of Π .

Example 3.6 Consider the following programs Π_1 , Π_4 and Π_5 .

$\Pi_1 :$	$a \leftarrow \text{not} b$	$\Pi_4 :$	$a \leftarrow \text{not} b$	$\Pi_5 :$	$a \leftarrow \text{not} b$
	$b \leftarrow \text{not} a$		$b \leftarrow \text{not} a$		$b \leftarrow \text{not} a$
	$d \leftarrow a$		$c \leftarrow \text{not} a, \text{not} b$		$d \leftarrow \text{not} a$
	$d \leftarrow b$				$d \leftarrow \text{not} b$

Let $H_1 = \langle \{d, a\}, \{b\} \rangle$, $H_2 = \langle \{d, b\}, \{a\} \rangle$, $H_3 = \langle \{b\}, \{a, c\} \rangle$ and $H_4 = \langle \{a\}, \{b, c\} \rangle$. H_1 and H_2 are L_m -preferable of Π_1 and Π_5 . H_3 and H_5 are L_m -preferable of Π_4 .

Proposition 3.11 *An L_m -preferable hypothesis tuple of a program Π is also L_m -saturated of Π , but the reverse is not true.*

Proof: Let H be an L_m -preferable hypothesis tuple of a program Π . The result follows immediately from Proposition 3.10 and the the maximality (wrt F-ordering) of H . Consider the program Π_5 in Example 3.6. $\langle \emptyset, \emptyset \rangle$ is L_m -saturated of Π_5 , but it is not L_m -preferable. $\square$

Definition 3.12 Let Π be a program. An $\mathcal{L}$ -admissible hypothesis tuple $H = \langle P, N \rangle$ of Π is said to be $\mathcal{L}$ -stable of Π if P and N partition the set δ_Π of all possible hypotheses and P contains no pairs of complementary literals a and $\neg a$.

Proposition 3.12 *An L_m -stable hypothesis tuple is L_m -preferable. But the reverse is not true.*

Proof: Let $H = \langle P, N \rangle$ be L_m -stable of a program Π . By Lemma 3.9, for any L_m -admissible $H' = \langle P', N' \rangle$, P' and N' are disjoint. Therefore, $H \subseteq H'$ implies $H = H'$. Consider the program $\{a \leftarrow \text{not} b, b \leftarrow \text{not} c, c \leftarrow \text{not} a, \}$. $\langle \emptyset, \emptyset \rangle$ is L_m -preferable, but not L_m -stable. $\square$

3.3 Fixed Point Characterization of $\mathcal{L}$ -Saturated Semantics

Fixed point structures of monotonic mappings on lattices play very important roles in the study of semantics for logic programming [3, 14, 18]. There was broad interest in the fixed point structure of monotonic mappings on complete semilattices [47, 48]. Using a similar approach, we apply fixed point structures on a complete semilattice to logic programming. The framework based on hypothetical reasoning can be structured as a complete semilattice consisting of admissible tuples. We associate with each program a monotonic mapping on the complete semilattice. All the semantics induced from saturated tuples are characterized as fixed points of the monotonic mapping on the complete semilattice. Furthermore, the set of these fixed points constitutes a complete semilattice itself.

3.3.1 Lattice and Fixed Points

This section introduces the requisite concepts and results concerning monotonic mappings and fixed points.

Definition 3.13 Let S be a set. An n -ary relation on S is a subset of S^n . When n is 2, the relation is a binary relation.

For a binary relation R , we denote the fact that $(x, y) \in R$ by xRy . Hereafter, a binary relation is abbreviated to a relation.

Definition 3.14 A relation $\leq$ on a set is a *partial order* if it satisfies the following conditions:

- (a) Reflexivity: $x \leq x$, for all $x \in S$;
- (b) Transitivity: $x \leq y$ and $y \leq z$ implies $x \leq z$, for all $x, y, z \in S$;
- (c) Antisymmetry: $x \leq y$ and $y \leq x$ implies $x = y$, for all $x, y \in S$.

$(S, \leq)$ is then called a *partially ordered set*.

We say x is smaller than y , or is below y when $x \leq y$; and x is larger than y , or is above y when $x \geq y$.

Definition 3.15 A partial order $\leq$ on a set S is a total order if it satisfies either $x \leq y$ or $y \leq x$, for all $x, y \in S$. $(S, \leq)$ is then called a *totally ordered set*.

Example 3.7 Both $(\mathcal{I}_L, \preceq)$ and $(\mathcal{I}_L, \subseteq)$ are partially ordered sets, where $\preceq$ and $\subseteq$ are the two natural orderings defined in Definition 2.33 on the set $\mathcal{I}_L$ of partial interpretations of a language L .

Definition 3.16 Let $(S, \leq)$ be a partially ordered set and $X \subseteq S$. $u \in S$ is called an *upper bound* of X if $x \leq u$, for all $x \in X$. Correspondingly, $l \in S$ is called a *lower bound* of X if $l \leq x$, for all $x \in X$.

Definition 3.17 Let $(S, \leq)$ be a partially ordered set and $X \subseteq S$. $u \in S$ is called a *least upper bound* of X if u is an upper bound of X and for all upper bounds u' of X , $u \leq u'$. Correspondingly, $l \in S$ is called a *greatest lower bound* of X if l is a lower bound of X and for all lower bounds l' of X , $l' \leq l$.

Both the least upper bound and the greatest lower bound of X are unique, if they exist, and we denote them by $\text{lub}(X)$ and $\text{glb}(X)$ respectively.

Definition 3.18 Let $(S, \leq)$ be a partially ordered set. $(S, \leq)$ is called a *complete lattice* if both $\text{lub}(X)$ and $\text{glb}(X)$ exist for every subset X of S .

We use $\top$ to denote the *top element* $\text{lub}(S)$ of a complete lattice $(S, \leq)$, and $\perp$ to denote the *bottom element* $\text{glb}(S)$ of a complete lattice $(S, \leq)$.

Example 3.8 Let $\mathcal{I}_{\mathbf{L}}$ be the set of all partial interpretations of a language $\mathbf{L}$. Then $(\mathcal{I}_{\mathbf{L}}, \subseteq)$ is a complete lattice, where $\subseteq$ is the F-ordering. For any subset S of $\mathcal{I}_{\mathbf{L}}$, the least upper bound is the union of every element in S , and the greatest lower bound is the intersection of every element in S .

Definition 3.19 Let $(S, \leq)$ be a partially ordered set and $\Phi : S \mapsto S$ be a mapping. Φ is *monotonic* if $\Phi(x) \leq \Phi(y)$, whenever $x \leq y$.

Definition 3.20 Let $(S, \leq)$ be a partially ordered set and $\Phi : S \mapsto S$ be a mapping. $x \in S$ is a *fixed point* of Φ if $\Phi(x) = x$.

Our interest in these definitions arises from the fact that the set of all partial interpretations forms a complete lattice with the F-ordering $\subseteq$, which is a partial order, and that for all the semantics reviewed below, every partial interpretation in the semantics can be characterized as a fixed point of a monotonic mapping defined on the complete lattice.

Definition 3.21 Let $(S, \leq)$ be a partially ordered set and $\Phi : S \mapsto S$ be a mapping. $l \in S$ is called the *least fixed point* of Φ if l is a fixed point of Φ and for all fixed points l' of Φ , $l \leq l'$. Correspondingly, we define *greatest fixed point* of Φ .

The following three propositions, Proposition 3.13, 3.14 and 3.15, are taken from [42]. Proposition 3.13 is attributed to Tarski [86].

Proposition 3.13 [42, Proposition 5.1] Let $(S, \leq)$ be a complete lattice and $\Phi : S \mapsto S$ be a monotonic mapping. Then Φ has a least fixed point, $\text{lfp}(\Phi)$, and a greatest fixed point, $\text{gfp}(\Phi)$. Furthermore, $\text{lfp}(\Phi) = \text{glb}(\{x : \Phi(x) = x\}) = \text{glb}(\{x : \Phi(x) \leq x\})$ and $\text{gfp}(\Phi) = \text{lub}(\{x : \Phi(x) = x\}) = \text{lub}(\{x : x \leq \Phi(x)\})$.

Definition 3.22 Let $(S, \leq)$ be a complete lattice and $\Phi : S \mapsto S$ be a monotonic mapping. Then the ordinal powers of Φ are defined as follows:

$$\begin{aligned} \Phi \uparrow^0 &= \perp, \\ \Phi \uparrow^{\alpha+1} &= \Phi(\Phi \uparrow^\alpha), \\ \Phi \uparrow^\alpha &= \text{lub}(\{\Phi \uparrow^\beta : \beta < \alpha\}), \text{ for limit ordinals } \alpha. \\ \Phi \downarrow^0 &= \top, \\ \Phi \downarrow^{\alpha+1} &= \Phi(\Phi \downarrow^\alpha), \\ \Phi \downarrow^\alpha &= \text{glb}(\{\Phi \downarrow^\beta : \beta < \alpha\}), \text{ for limit ordinals } \alpha. \end{aligned}$$

The next result is a well-known characterization of the least fixed point and the greatest fixed point in terms of ordinal powers of Φ .

Proposition 3.14 [42, Proposition 5.3] *Let $(S, \leq)$ be a complete lattice and $\Phi : S \mapsto S$ be a monotonic mapping. Then, for any ordinal α , $\Phi \uparrow^\alpha \leq \text{lfp}(\Phi)$ and $\Phi \downarrow^\alpha \geq \text{gfp}(\Phi)$. Furthermore, there exist ordinals β_1 and β_2 such that $\gamma_1 \geq \beta_1$ implies $\Phi \uparrow^{\gamma_1} = \text{lfp}(\Phi)$ and $\gamma_2 \geq \beta_2$ implies $\Phi \downarrow^{\gamma_2} = \text{gfp}(\Phi)$.*

The least fixed point of Φ can be further characterized by the ordinal power of ω , which is the set of all finite ordinals, when Φ satisfies further condition.

Definition 3.23 Let $(S, \leq)$ be a partially ordered set and $X \subseteq S$. X is said to be *directed* if every finite subset of X has an upper bound in X .

Definition 3.24 Let $(S, \leq)$ be a complete lattice and $\Phi : S \mapsto S$ be a mapping. Φ is said to be *continuous* if $\Phi(\text{lub}(X)) = \text{lub}(\Phi(X))$, for every directed subset X of S .

Every continuous mapping is monotonic. However, the converse is not true.

The next result is usually attributed to Kleen.

Proposition 3.15 [42, Proposition 5.4] *Let $(S, \leq)$ be a complete lattice and $\Phi : S \mapsto S$ be a continuous mapping. Then $\text{lfp}(\Phi) = \Phi \uparrow^\omega$.*

We now turn to the notion of *complete semilattice*.

Definition 3.25 Let $(S, \leq)$ be a partially ordered set. $(S, \leq)$ is a *complete semilattice* if

- (a) $\text{glb}(X)$ exists for every nonempty subset X of S ,
- (b) $\text{lub}(X)$ exists for every nonempty directed subset X of $(S, \leq)$.

Proposition 3.16 [18, Proposition 6.2] *Let $(S, \leq)$ be a complete semilattice, and $\Phi : S \mapsto S$ is monotonic.*

- (a) *If $x \leq \Phi(x)$, then Φ has a maximal fixed point larger than x .*
- (b) (1) *If $x \leq \Phi(x)$, then Φ has a smallest fixed point larger than x .*
 (2) *If $x \leq \Phi(x)$, $x \leq y$, and $\Phi(y) \leq y$, then the smallest fixed point larger than x will be smaller than y .*
- (c) *If $\Phi(y) \leq y$, then Φ has a largest fixed point smaller than y .*

In a complete lattice, Proposition 3.13 guarantees that every monotonic mapping has a least and a largest fixed point. And the collection of the fixed points constitutes a complete lattice [86]. Since a complete semilattice $(S, \leq)$ must contain a least element, (a) and (b) of Proposition 3.16 guarantee the existence of a least and a maximal fixed point of a monotonic mapping. There need not be a largest element of S , so (c) cannot be used to conclude the existence of a largest fixed point. Though a monotonic mapping may not have a largest fixed point, it has several maximal ones instead. Our interest in the fixed point structure of monotonic mappings on complete semilattices arises from the facts that the collection of the admissible tuples of a logic program constitutes a complete semilattice, the saturated tuples are fixed points of a monotonic mapping associated with the program on the complete semilattice, and furthermore, the collection of saturated tuples also constitutes a complete semilattice.

Proposition 3.17 [18, Proposition 6.4] *Let $(S, \leq)$ be a complete semilattice, and $\Phi : S \mapsto S$ is monotonic.*

(a) *Suppose $x \leq \Phi(x)$.*

(1) *The following defines a sequence of elements $\Phi \uparrow^\alpha (x)$ of S for each ordinal α :*

$$\begin{aligned}\Phi \uparrow^0 (x) &= x, \\ \Phi \uparrow^{\alpha+1} (x) &= \Phi(\Phi \uparrow^\alpha (x)), \\ \Phi \uparrow^\alpha &= \text{lub}(\{\Phi \uparrow^\beta (x) : \beta < \alpha\}), \text{ for limit ordinals } \alpha.\end{aligned}$$

(2) *The sequence $\Phi \uparrow^\alpha (x)$ increases with α and converges to the least fixed point of Φ larger than x .*

(c) *Suppose $\Phi(x) \leq x$.*

(1) *The following defines a sequence of elements $\Phi \downarrow^\alpha (x)$ of S for each ordinal α :*

$$\begin{aligned}\Phi \downarrow^0 (x) &= x, \\ \Phi \downarrow^{\alpha+1} (x) &= \Phi(\Phi \downarrow^\alpha (x)), \\ \Phi \downarrow^\alpha &= \text{glb}(\{\Phi \downarrow^\beta (x) : \beta < \alpha\}), \text{ for limit ordinals } \alpha.\end{aligned}$$

(2) *The sequence $\Phi \downarrow^\alpha (x)$ decreases as α increases and converges to the largest fixed point of Φ smaller than x .*

Let $\perp$ be the least element of S . Trivially $\perp \leq \Phi(\perp)$, so by (a) the sequence $\Phi \uparrow^\alpha (\perp)$ converges to the least fixed point of Φ . The least ordinal α for which $\Phi \uparrow^\alpha (\perp)$ is the least fixed point of Φ is called the *closure ordinal* of Φ .

Definition 3.26 Let $(S, \leq)$ be a partially ordered set and $X \subseteq S$. $m \in X$ is called a maximal element of X if X contains no larger element, i.e. $m \leq x$ implies $m = x$ for all $x \in X$. Correspondingly, $m \in X$ is called a minimal element of X if X contains no smaller element, i.e. $x \leq m$ implies $x = m$ for all $x \in X$.

A criterion for the existence of maximal elements in a partially ordered set is given by the maximal principle, which goes back to Kuratowski, Hausdorff, and Zorn and is often called *Zorn's Lemma*. There are several equivalent statements; we assume the following:

Proposition 3.18 (*Zorn's Lemma*. [57, p 198]) Let $(S, \leq)$ be a partially ordered set. If every totally ordered subset $(X, \leq)$, where $X \subseteq S$, has an upper bound, then S contains a maximal element.

3.3.2 Fixed Point Characteristics of $\mathcal{L}$ -Saturated Hypothesis Tuples

In this section, we will discuss some structural properties of admissible hypothesis tuples and saturated hypothesis tuples and introduce the concept of saturated closure.

In the following, Π is a program, AH_{L_m} denotes the set of all L_m -admissible hypothesis tuples of Π .

Lemma 3.19 Let DS be a nonempty directed subset of AH_{L_m} . Then $UH = \cup_{H \in DS} H$ is L_m -admissible of Π .

Proof: We first prove the following fact:

For any ground literal l , $UH \uplus \Pi \models_{L_m} l$ iff there exists an element $H \in DS$ such that $H \uplus \Pi \models_{L_m} l$.

One direction is trivial by Proposition 3.5. Suppose $UH \uplus \Pi \models_{L_m} l$. By the compactness of L_m , there exists a finite subset FSP of $UH \uplus \Pi$ such that $FSP \models_{L_m} l$. Then there exists a finite subset FS of UH such that $UFS \uplus \Pi \models_{L_m} l$, where $UFS \cup_{H \in FS} H$. Because DS is directed and FS is a finite subset of DS , there exists an upper bound H' of FS in DS . Clearly $UFS \subseteq H'$. By Proposition 3.5, $H' \uplus \Pi \models_{L_m} l$.

We then prove that UH is L_m -noncontradictory. If it is not, there exists $l \in UH$ such that $UH \uplus \Pi \models_{L_m} \neg l$ and there exists an element $H_l \in DS$ such that $l \in H_l$. By the above fact, there exists some $H \in DS$ such that $H \uplus \Pi \models_{L_m} \neg l$. Let H' be the least upper bound of $\{H, H_l\}$. By Proposition 3.5, $H' \uplus \Pi \models_{L_m} \neg l$. Since $l \in H'$, H' is not L_m -noncontradictory. This contradicts $H' \in AH_{L_m}$.

We then prove UH is L_m -founded. Let $UH = \langle UP, UN \rangle$. For each $l \in UP$, there exists some $H_l = \langle P_l, N_l \rangle \in DS$ such that $l \in P_l$. Then $H_l \uplus \Pi \models_{L_m} l$. It follows from Proposition 3.5 that $UH \uplus \Pi \models_{L_m} l$. So UH is positively L_m -founded. For each $l \in UN$, there exists some $H_l = \langle P_l, N_l \rangle \in DS$ such that $l \in N_l$. l is L_m -false-assumable wrt H_l . By Proposition 3.8, l is L_m -false-assumable wrt UH since $H_l \subseteq UH$. Therefore, UH is L_m -admissible. $\square$

Theorem 3.20

- (a) $(AH_{L_m}, \subseteq)$ is a complete semilattice;
- (b) For every L_m -admissible tuple $H \in AH_{L_m}$, there exists at least one L_m -preferable tuple $H_p \in AH_{L_m}$ such that $H \subseteq H_p$.

Proof: Proof of (a): Let SS be a nonempty subset of AH_{L_m} , and SLB be the set of lower bounds of SS , i.e. $SLB = \{H : H \subseteq \cap_{K \in SS} K, \text{ and } H \in AH_{L_m}\}$. We prove that $GLB = \cup_{H \in SLB} H$ is the greatest lower bound of SS . Clearly, $GLB \subseteq K$ for each $K \in SS$, and $H \subseteq GLB$ for each $H \in SLB$. Then we need only to show that $GLB \in AH_{L_m}$. Let $GLB = \langle GP, GN \rangle$. We show GLB is L_m -admissible of Π .

Since there exists $K \in SS$ such that $GLB \subseteq K$ and K is L_m -noncontradictory, it follows from Proposition 3.6 that GLB is L_m -noncontradictory. We now prove that GLB is L_m -founded. For each literal $l \in GP$, there exists some $H_l = \langle P_l, B_l \rangle \in SLB$ such that $l \in P_l$ and $H_l \uplus \Pi \models_{L_m} l$. By Proposition 3.5, $GLB \uplus \Pi \models_{L_m} l$ since $H_l \subseteq GLB$. Therefore, GLB is positively L_m -founded. For each literal $l \in GN$, there exists some $H_l = \langle P_l, B_l \rangle \in SLB$ such that $l \in N_l$ and l is L_m -false-assumable wrt H_l . By Proposition 3.8, l is L_m -false-assumable wrt GLB since $H_l \subseteq GLB$. Therefore, GLB is negatively L_m -founded. GLB is L_m -admissible and is indeed the greatest lower bound of SS .

Let DS be a nonempty directed subset of AH_{L_m} , and $LUB = \cup_{H \in DS} H$. Then LUB is a least upper bound of DS since LUB is L_m -admissible of Π by Lemma 3.19, i.e. $LUB \in AH_{L_m}$.

Proof of (b): For any element $H \in AH_{L_m}$, let $UH = \{H' \in AH_{L_m} : H \subseteq H'\}$. Since $H \in UH$, UH is not empty. We claim that every totally ordered subset S of UH has an upper bound in UH . Let $US = \cup_{H' \in S} H'$. By Lemma 3.19, US is L_m -admissible. Clearly, US is an upper bound of S and $US \in UH$. Since UH is a nonempty partial ordered set in which every totally ordered subset has an upper bound, by Proposition 3.18 (Zorn's Lemma), there exists a maximal element PH in UH . By the definition of UH , $H \subseteq PH$ and PH is also a maximal element of AH_{L_m} . $\square$

Example 3.9 Consider the following program Π_1 and Π_5 :

$\Pi_1 :$	$a \leftarrow \text{not} b$	$\Pi_5 :$	$a \leftarrow \text{not} b$
	$b \leftarrow \text{not} a$		$b \leftarrow \text{not} a$
	$d \leftarrow a$		$d \leftarrow \text{not} a$
	$d \leftarrow b$		$d \leftarrow \text{not} b$

$H_1 = \langle \{d, a\}, \{b\} \rangle$, $H_2 = \langle \{d, b\}, \{a\} \rangle$ and $\langle \emptyset, \emptyset \rangle$ are L_3 -admissible of both Π_1 and Π_5 . Among them, H_1 and H_2 are L_m -preferable. $H_4 = \langle \{d\}, \emptyset \rangle$ is L_2 -admissible of Π_1 , but not of Π_5 . H_1 and H_2 are also L_2 -preferable of Π_1 and Π_5 and H_3 is also L_2 -admissible of Π_5 .

Example 3.9 is a counterexample that $(AH_{L_m}, \subseteq)$ is not a supremum semilattice¹ because the two L_m -preferable sets H_1 and H_2 do not have a least upper bound.

Now we introduce the definition of saturated closure of an admissible hypothesis tuple.

Definition 3.27 Let H be $\mathcal{L}$ -admissible of a program Π . The $\mathcal{L}$ -saturated closure of H is defined as the $\subseteq$ -least $\mathcal{L}$ -saturated hypothesis tuple H^* such that $H \subseteq H^*$.

That is, if H^* is the $\mathcal{L}$ -saturated closure of H , then for any $\mathcal{L}$ -saturated tuple J , $H^* \subseteq J$ whenever $H \subseteq J$.

Let $AH_{\mathcal{L}}$ be the set of all $\mathcal{L}$ -admissible hypothesis tuples of a program Π . By Theorem 3.20, we showed that $(AH_{L_m}, \subseteq)$ is a complete semilattice. In the following, we associate an operator with each program. This operator turns out to be monotonic on $AH_{\mathcal{L}}$ and characterizes saturated closures as its fixed points.

Definition 3.28 For a program Π , $\Psi_{\Pi} : AH_{\mathcal{L}} \mapsto AH_{\mathcal{L}}$ is a mapping defined on the complete semilattice $(AH_{\mathcal{L}}, \subseteq)$ as follows:

$$\Psi_{\Pi}(H) = \langle T, F \rangle,$$

where $T = \{l : l \text{ is positively } \mathcal{L}\text{-founded wrt } H\}$ and $F = \{l : l \text{ is } \mathcal{L}\text{-false-assumable wrt } H\}$.

It follows from the L_m -foundedness of H that $H \subseteq \Psi_{\Pi}(H)$. By Proposition 3.10, $\langle T, F \rangle$ is L_m -admissible of Π . The operator Ψ preserves the L_m -admissibility of hypothesis tuples. It follows that Ψ_{Π} maps elements of AH_{L_m} to elements of AH_{L_m} .

Proposition 3.21 Ψ_{Π} is monotonic when $\mathcal{L}$ is L_m .

¹A partially ordered set $(S, \leq)$ is called a *supremum semilattice* if $\text{lub}(X)$ exists for every finite subset X of S .

Proof: The result follows directly from Proposition 3.5 and Proposition 3.8. $\square$

The next proposition shows that L_m -saturated sets can be characterized as fixed points of Ψ_Π .

Proposition 3.22 *Let $H \in AH_{L_m}$ be L_m -admissible of Π . H is L_m -saturated of Π iff H is a fixed point of Ψ_Π , i.e. $H = \Psi_\Pi(H)$.*

Proof: The result follows from Definition 3.10 and 3.28. $\square$

Next we show that the L_m -saturated closure of an L_m -admissible hypothesis tuple H always exists and it can be constructed from H as an ordinal power of Ψ_Π and characterized as a fixed point of Ψ_Π .

Definition 3.29 Let $H \in AH_{L_m}$. Let

$$\begin{aligned} H_0 &= H; \\ H_{\alpha+1} &= \Psi_\Pi(H_\alpha), \\ H_\alpha &= \bigcup_{\beta < \alpha} H_\beta, \text{ for limit ordinals } \alpha. \end{aligned}$$

Proposition 3.23 *$\{H_\alpha\}$ is a sequence of elements of AH_{L_m} .*

Proof: We prove by transfinite induction that $H_\gamma \subseteq H_\alpha$ whenever $\gamma < \alpha$ and H_α is L_m -admissible. Assume for every ordinal $\beta < \alpha$, the fact is true of β . If $\alpha \neq 0$ is a limit, $(\{H_\beta : \beta < \alpha\}, \subseteq)$ is a totally ordered set, so $\{H_\beta : \beta < \alpha\}$ is a directed subset of AH_{L_m} . By Lemma 3.19, $\bigcup_{\beta < \alpha} H_\beta$ is L_m -admissible. (When α is 0, the claim is vacuously true.) Since Ψ_Π maps elements of AH_{L_m} to elements of AH_{L_m} and $I \subseteq \Psi_\Pi(I)$ for any $I \in AH_{L_m}$, $H_{\alpha+1}$ is L_m -admissible and $H_\alpha \subseteq H_{\alpha+1}$. Induction is completed. $\square$

By Proposition 3.23, H_α is the least upper bound of $\{H_\beta : \beta < \alpha\}$ when α is a limit ordinal. Recall the sequence defined in Proposition 3.17, clearly

$$H_\alpha = \Psi \uparrow^\alpha (H) \text{ for any ordinal } \alpha.$$

By Proposition 3.17, the transfinite sequence $\{H_\alpha\}$ increases with α and converges to the least fixed point of Ψ larger than H . We denote this fixed point by H_{fix} . Since the Herbrand base B_L is countable, so is δ_Π . There exists a smallest countable ordinal δ such that $H_{\delta+1} = \Psi_\Pi(H_\delta) = H_\delta = H_{fix}$.

Proposition 3.24 *Let H be L_m -admissible. H_{fix} is the L_m -saturated closure of H .*

Proof: Since H_{fix} is the least fixed point of Ψ_Π larger than H , it is the L_m -saturated closure of H by Proposition 3.22. $\square$

Proposition 3.24 indicates that with the introduction of the monotonic operator Ψ_Π , we can construct the $\mathcal{L}$ -saturated closure of an L_m -admissible hypothesis tuple H through the ordinal power of Ψ_Π on H . In the following, we discuss the $\mathcal{L}$ -saturated closure of the least element $\langle \emptyset, \emptyset \rangle$ of $(AH_{\mathcal{L}}, \subseteq)$. It turns out to be the least $\mathcal{L}$ -saturated hypothesis tuple (when $\mathcal{L}$ is L_m).

Definition 3.30 Let $H_\perp = \langle \emptyset, \emptyset \rangle$ denote the least element of $(AH_{\mathcal{L}}, \subseteq)$. The $\mathcal{L}$ -saturated closure $H_\perp^*$ of $H_\perp$ is called the $\mathcal{L}$ -well-founded hypothesis tuple of Π . $H_\perp^*$ is also denoted as $H_{\mathcal{L}-wf}$.

Proposition 3.25 H_{L_m-wf} is the least fixed point $lfp(\Psi_\Pi)$ of Ψ_Π and is L_m -saturated of Π .

Proof: It follows from Proposition 3.17 and Proposition 3.22. $\square$

The set of all L_m -saturated hypothesis tuples of a program Π has similar structural properties to the properties of the set of all L_m -admissible hypothesis tuples as indicated by Theorem 3.20.

Theorem 3.26 Let SH_{L_m} be the set of all L_m -saturated hypothesis tuples of a program Π .

- (a) $(SH_{L_m}, \subseteq)$ is a complete semilattice;
- (b) H_{L_m-wf} is the least element of $(SH_{L_m}, \subseteq)$.

Proof:

Proof of (a): Let SS be a nonempty subset of SH_{L_m} . Let SLB be the set of lower bounds of SS , i.e. $SLB = \{H : H \subseteq \bigcap_{K \in SS} K, \text{ and } H \in SH_{L_m}\}$. Let $GLB = \langle GP, GN \rangle = \bigcup_{H \in SLB} H$. We first prove that GLB is L_m -admissible.

For any $K \in SS$, $GLB \subseteq K$. It follows from Proposition 3.6 that GLB is L_m -noncontradictory because K is. For each literal $l \in GP$, there exists some $H_l = \langle P_l, N_l \rangle \in SLB$ such that $l \in P_l$ and $H_l \uplus \Pi \models_{L_m} l$. By Proposition 3.5, $GLB \uplus \Pi \models_{L_m} l$ since $H_l \subseteq GLB$. Therefore, GLB is positively L_m -founded. For each literal $l \in GN$, there exists some $H_l = \langle P_l, N_l \rangle \in SLB$ such that $l \in N_l$ and l is L_m -false-assumable wrt H_l . By Proposition 3.8, l is L_m -false-assumable wrt GLB since $H_l \subseteq GLB$. Therefore, GLB is negatively L_m -founded. Thus we have proved that GLB is L_m -admissible. By Proposition 3.24, the L_m -saturated closure GLB^* of GLB is GLB_{fix} . Because $GLB \subseteq GLB_{fix}$, GLB^* is an upper bound of SLB by Proposition 3.21.

It follows from Definition 3.27 that GLB^* is a lower bound of SS since for every $K \in SS$, $GLB \subseteq K$ and K is L_m -saturated.

Let DS be a nonempty directed subset of SH_{L_m} , and $LUB = \cup_{H \in DS} H$. By Lemma 3.19, LUB is L_m -admissible. Then by Proposition 3.24, the L_m -saturated closure LUB^* of LUB is LUB_{fix} . LUB^* is an upper bound of DS because $LUB \subseteq LUB^*$. For any upper bound $UB \in SH_{L_m}$ of DS , $LUB \subseteq UB$. Because Ψ_Π is monotonic by Proposition 3.21, $LUB_{fix} \subseteq UB_{fix}$. Because UB is L_m -saturated, $UB = UB_{fix}$. Hence, $LUB^* \subseteq UB$ and LUB^* is the least upper bound of DS .

Proof of (b): By Proposition 3.22, every L_m -saturated hypothesis tuple is a fixed point of the operator Ψ_Π . Since H_{L_m-wf} is the least fixed point of Ψ_Π by Proposition 3.25, it is the least element of SH_{L_m} . $\square$

3.4 A Framework for Semantics of Disjunctive Logic Programs

The framework in Section 3.2 is proposed for extended programs. This section extends the framework to the disjunctive extended programs defined below. The approach used to extend the framework is believed to be first proposed here, and is so general that it can be readily used in other formalisms.

Definition 3.31 A *disjunctive* logic program of a language L is defined as a general logic program such that the head of each of its clauses is a disjunction of literals $K_1 \vee \dots \vee K_l$, $l \geq 1$. A program clause with such a head is also said to be disjunctive.

Clearly, an extended program is a special form of a disjunctive program.

A program clause in a disjunctive program is conventionally denoted as

$$K_1, \dots, K_l \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_{m+n},$$

where $l \geq 1$.

Definition 3.32 Let Π be a disjunctive program and $H = \langle P, N \rangle \subseteq \Delta_\Pi$. A *semantical instance* π of Π wrt H is a set obtained from $[\Pi]_L$ by replacing each disjunctive clause

$$K_1, \dots, K_l \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_{m+n},$$

where $l \geq 1$, in $[\Pi]_L$ with an extended clause

$$K_i \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_{m+n},$$

where K_i , $1 \leq i \leq l$ satisfies the following

- (a) $K_i \in P$ if there exists $K_j \in P$, $1 \leq j \leq l$; and
- (b) $K_i \notin N$ if there exists $K_j \notin N$, $1 \leq j \leq l$.

In the above definition, the head of a disjunctive clause is replaced with a literal K_i , which is a disjunct of the head. K_i is chosen from P whenever possible; and K_i is not to be chosen from N unless it has to be. We adopt the exclusive reading of a disjunction head in logic programming, i.e. one and only one disjunct K_i is to be derived from the clause

$$K_1, \dots, K_l \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_{m+n}.$$

Then, a semantical instance can be used to justify the acceptability of a hypothesis tuple. When the program Π is not disjunctive, it has only a unique semantical instance wrt any hypothesis tuple H . Clearly, a semantical instance is not disjunctive. Hence, all the definitions and concepts introduced in Section 3.2 for extended programs can be naturally extended to disjunctive programs.

Definition 3.33 Let Π be a disjunctive program and $H \subseteq \Delta_\Pi$. H is said to be $\mathcal{L}$ -admissible ($\mathcal{L}$ -saturated, $\mathcal{L}$ -preferable, $\mathcal{L}$ -stable or $\mathcal{L}$ -well-founded) of Π if H is $\mathcal{L}$ -admissible ($\mathcal{L}$ -saturated, $\mathcal{L}$ -preferable, $\mathcal{L}$ -stable or $\mathcal{L}$ -well-founded) of each semantical instance π of Π wrt H .

With Definition 3.33, the framework of Section 3.2 is extended to disjunctive programs.

It should be noted that it is essential to consider each of the semantical instances of a disjunctive program Π , rather than consider the set of all semantical instances of clauses in Π .

Example 3.10 Consider a program $\Pi : a \vee b$. Let $H_1 = \langle \{a\}, \{b\} \rangle$ and $H_2 = \langle \{b\}, \{a\} \rangle$. $\pi_1 : \{a\}$ is the unique semantical instance of Π wrt H_1 , and $\pi_2 : \{b\}$ is the unique semantical instance of Π wrt H_2 . H_1 and H_2 are L_m -stable of π_1 and π_2 respectively. However, neither H_1 nor H_2 is positively L_m -founded of $\pi_3 : \{a, b\}$. It should be noted that H_1 and H_2 correspond to two minimal Herbrand model of Π .

Definition 3.32 suggests that the intuitive reading of a disjunctive clause

$$K_1, \dots, K_l \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_{m+n},$$

is that one and only one disjunct K_i in the head is to be derived from this clause if the body has been derived. (Of course, more than one disjunct may be derived from the whole program due to other clauses in the program.) Such an intuitive

reading has been adopted in most existing semantics for disjunctive programming and indefinite databases, including minimal model semantics [75, 58] and 3-valued stable model semantics [73]. Definition 3.32 first syntactically formalizes the intuitive reading of disjunctive programs, and it also postulates a criterion for a semantical model of a disjunctive program; that a semantical model of a disjunctive program should also serve as a semantical model of each semantical instance of the program wrt the semantical model.

Example 3.11 Suppose a unit clause a is added to the program Π in Example 3.10. Consider the revised program $\Pi' : a \vee b; a$. Then $H_1 = \langle \{a\}, \{b\} \rangle$ is still to be considered a semantical model since $\pi_1 : \{a\}$ is the unique semantical instance of Π' wrt H_1 . Let $H_3 = \langle \{a, b\}, \emptyset \rangle$. Then $\pi_3 : \{a; b\}$ is also a semantical instance of Π' wrt H_3 . Although H_3 is positively L_m -founded of $\pi_3 : \{a, b\}$, it is not so of π_1 . It should be noted that H_1 is a minimal Herbrand model of Π' while H_3 is not.

3.5 Semantical Extensions Based on $\mathcal{L}$ -Saturation

With the introduction of admissible, saturated and preferable hypotheses, we can define the corresponding concepts of scenarios, explanations and extensions with Poole's framework.

Definition 3.34 Let Π be a (disjunctive) program. An $\mathcal{L}$ -admissible scenario of (Π, Δ_Π) is defined as $H \uplus \pi$, where $H \subseteq \Delta_\Pi$ is $\mathcal{L}$ -admissible of Π , and π is a semantical instance of Π wrt H . An $\mathcal{L}$ -admissible scenario is said to be $\mathcal{L}$ -saturated ($\mathcal{L}$ -preferable, $\mathcal{L}$ -stable, $\mathcal{L}$ -well-founded) if H is $\mathcal{L}$ -saturated ($\mathcal{L}$ -preferable, $\mathcal{L}$ -stable, $\mathcal{L}$ -well-founded) of Π .

The extensions of a reasoning system $(\mathcal{F}, \Delta)$ are defined as sets of logical consequences of $\mathcal{F}$ and maximal sets (wrt set inclusion) of hypotheses which preserve consistency of the reasoning system. In the context of logic programming, consistency is always preserved due to the noncontradictoriness of admissible hypothesis tuples as indicated by Proposition 3.7. We are more interested in maximizing hypothesis tuples wrt the foundedness (not wrt set inclusion) and preserving the admissibility of the hypotheses. Consequently, the corresponding concept of extensions is defined as sets of logical consequences of a program Π incorporating a saturated hypothesis tuple, which is maximized wrt the foundedness and preserves the admissibility.

Definition 3.35 Let Π be a program. An $\mathcal{L}$ -saturated extension of (Π, Δ_Π) is the set of logical consequences of an $\mathcal{L}$ -saturated scenario of (Π, Δ_Π) . The extension is

said to be an $\mathcal{L}$ -preferable extension ($\mathcal{L}$ -well-founded extension, $\mathcal{L}$ -stable extension) if the scenario is an $\mathcal{L}$ -preferable ($\mathcal{L}$ -well-founded, $\mathcal{L}$ -stable) scenario.

The next proposition shows that our framework for logic programming preserves the equivalence between explicability and being in a saturated extension.

Proposition 3.27 *A sentence S is explicable from (Π, Δ_Π) if and only if S is in some $\mathcal{L}$ -saturated extension of (Π, Δ_Π) .*

Proof: If S is in some L_m -saturated extension, then there exists an L_m -saturated scenario $H \uplus \pi$ such that $H \uplus \pi \models_{L_m} S$. Thus $H \uplus \pi$ is an explanation of S .

If S is explicable, there exists an L_m -admissible scenario $H \uplus \pi$ such that $H \uplus \pi \models_{L_m} S$. There exists an L_m -saturated hypothesis tuple H' such that $H \subseteq H'$ by Proposition 3.24. By Proposition 3.5, $H' \uplus \pi \models_{L_m} S$. Therefore, S is in an L_m -saturated extension. $\square$

Taking logic $\mathcal{L}$ as an underlying logic of programs, we can say that the set of all $\mathcal{L}$ -saturated scenarios represents the *universe of all possible semantics* (the set of all possible worlds). Among these semantics, the $\mathcal{L}$ -well-founded extension represents a skeptical point of view in knowledge representations. In such a semantics, only things which hold in all possible worlds are "believed" true. It is a skeptical semantics. On the other hand, the $\mathcal{L}$ -preferable extensions represent a credulous point of view in knowledge representations. In such a semantics, the possible worlds taken into account are the worlds where a reasoning agent can reach as many conclusions as possible from a program considered as an incomplete knowledge base. Each $\mathcal{L}$ -preferable scenario corresponds to such a possible world. It is a credulous semantics. The $\mathcal{L}$ -stable extensions represent an omniscient point of view since an $\mathcal{L}$ -stable tuple partitions all the possible hypotheses and thus everything is either assumed true or false.

It follows from Theorem 3.20 and Theorem 3.26 that L_m -saturated, L_m -preferable, L_m -well-founded semantics are defined for the class of all programs while L_m -stable semantics is defined only for the class of programs that have at least one $\mathcal{L}$ -stable scenario.

3.6 Impact of Classical Negation in Extended programs

An extended logic program is a set of formulas of the form

$$L_0 \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_{m+n},$$

where the L_i 's are literals, which may be of form $\neg a$. The classical negation $\neg$ in logic programming complicates matters in search for a proper semantics of extended programs. Consider the programs $\Pi_1 : \{a \leftarrow \neg b; \neg a \leftarrow\}$. In the programs, $\neg a$ is explicitly specified true. Should we infer that b is true? Since $\leftarrow$ embeds rule-like behavior, i.e. $K \leftarrow \dots$ is considered a definition of K , the answer to the question should be 'No'. However, the semantics in Section 3.5 would give a 'Yes' answer to the question.

One of major concerns in search of a reasonable declarative semantics of programs is the orientation bearing in the clause. This is the essential difference between logic programs with negative literals and "pure" non-Horn clauses. In this section, we revise the framework proposed in Section 3.2 and 3.4 to reflect the rule-like behavior of $\leftarrow$ even in the presence of classical negation $\neg$.

3.6.1 Proof by Cases and Oriented Consequences

The logical implication $\models_{\mathcal{L}}$ treats $K \leftarrow \dots$ as a logical sentence of the logic $\mathcal{L}$, not as a definition of K . So $\models_{\mathcal{L}}$ distorts the meaning embedded in the orientation of $\leftarrow$. However, the incorporation operator $\uplus$ preserves the information to a certain extent, and it works particularly well for normal programs. Given an extended program, that is, a set of program clauses with classical negation, which have embedded orientation, we may wish to find, at least theoretically, all program clauses that can be derived from the program without violating the oriented implication pattern.

Definition 3.36 Let Π be a program of a language $\mathbf{L}$. Then the oriented consequences of Π are clauses inductively defined as follows.

- **Basis.**

- (a) $l \leftarrow l$ is an oriented consequence of Π for every literal $l \in \text{Lit}_{\mathbf{L}}$;
- (b) Any instantiated clause $\alpha \in [\Pi]_{\mathbf{L}}$ of Π is an oriented consequence of Π .

- **Induction.**

- (a) Horn-type rule:

If both $\alpha : l \leftarrow k, l_1, \dots, l_n, \text{not} l_{n+1}, \dots, \text{not} l_{n+m}$ and
 $\beta : k \leftarrow k_1, \dots, k_r, \text{not} k_{r+1}, \dots, \text{not} k_{r+s}$ are oriented consequences of Π ,
 so is
 $\gamma : l \leftarrow l_1, \dots, l_n, k_1, \dots, k_r, \text{not} l_{n+1}, \dots, \text{not} l_{n+m}, \text{not} k_{r+1}, \dots, \text{not} k_{r+s}$

- (b) Proof-by-case rule:

If both $\alpha : l \leftarrow b, l_1, \dots, l_n, \text{not} l_{n+1}, \dots, \text{not} l_{n+m}$ and
 $\beta : l \leftarrow \neg b, k_1, \dots, k_r, \text{not} k_{r+1}, \dots, \text{not} k_{r+s}$ are oriented consequences, so

is

$$\gamma : l \leftarrow l_1, \dots, l_n, k_1, \dots, k_r, \text{not} l_{n+1}, \dots, \text{not} l_{n+m} \text{not} k_{r+1}, \dots, \text{not} k_{r+s}$$

Nothing else is an oriented consequence of Π . In the above, $l, k, l_i, k_i \in \text{Lit}_{\mathbf{L}}$ are literals. b and $\neg b$ may be the logical form of **not** $\neg a$ and **nota** respectively.

A similar definition was given in [80, 28] just for normal programs. The set of all oriented consequences, called derived rules in [80], is the set of all ground clauses that can be derived from Π without violating the oriented implication patterns of the program.

We write $\Pi \vdash_{\text{Case}} \alpha$ if α is an oriented consequences of a program Π . If α can be derived from Π without using the proof-by-case rule, we write $\Pi \vdash_{\text{Horn}} \alpha$.

Example 3.12 Consider the program Π_1 .

$$\begin{aligned} \Pi_1 : \quad & a \leftarrow \text{not} b \\ & b \leftarrow \text{nota} \\ & d \leftarrow a \\ & d \leftarrow b \end{aligned}$$

Besides every clause in Π_1 , $\Pi_1 \vdash_{\text{Horn}} d \leftarrow \text{nota}$ and $\Pi_1 \vdash_{\text{Horn}} d \leftarrow \text{not} b$. And $\Pi_1 \vdash_{\text{Case}} d$.

The concept of oriented consequences is applicable to any set of ground program clauses. Let S be a (finite or infinite) set of ground program clauses, we write $S \vdash_{\text{Case}} \alpha$ if α can be derived from S by the rules in Definition 3.36, and $S \vdash_{\text{Horn}} \alpha$ if the proof-by-case rule is not used. For a program Π , we would simply write $\Pi \vdash_{\text{Case}} \alpha$ and $\Pi \vdash_{\text{Horn}} \alpha$ for $[\Pi]_{\mathbf{L}} \vdash_{\text{Case}} \alpha$ and $[\Pi]_{\mathbf{L}} \vdash_{\text{Horn}} \alpha$ respectively.

Let L_2 and L_3 be Lukasiewiczian 2-valued and 3-valued logic respectively. We now show the relationship between $\models_{L_2}$ and $\vdash_{\text{Case}}$ and between $\models_{L_3}$ and $\vdash_{\text{Horn}}$.

Lemma 3.28 *Let S be a set of ground program clauses. Then for any program clause γ ,*

- (a) $S \models_{L_2} \gamma$ if $S \vdash_{\text{Case}} \gamma$;
- (b) $S \models_{L_3} \gamma$ if $S \vdash_{\text{Horn}} \gamma$;
- If S contains only normal program clauses, then for any ground literal $l \in \text{Lit}_{\mathbf{L}}$,*
- (c) $S \vdash_{\text{Case}} l$ iff $S \models_{L_2} l$;
- (d) $S \vdash_{\text{Horn}} l$ iff $S \models_{L_3} l$,

Proof: (a) and (b): Proof by structural induction,

Basis: The result follows vacuously from $\gamma \in S$.

Induction: Let α and β be as in Definition 3.36. Suppose $S \models_{L_m} \alpha$ and $S \models_{L_m} \beta$. Let M be a model of S .

Horn-type rule:

$/l/M \geq \min\{/k/M, /l_i/M, /\neg l_{n+j}/M\}$ and $/k/M \geq \min\{/k_i/M, /\neg k_{r+j}/M\}$ since $S \models_{L_m} \alpha, \beta$. Therefore, $/l/M \geq \min\{/l_i/M, /k_j/M, /\neg l_{n+l}/M, /\neg k_{r+k}/M\}$ and $/\gamma/M = \mathbf{t}$. Thus $S \models_{L_m} \gamma$.

Proof-by-case rule:

$/l/M \geq \min\{/b/M, /l_i/M, /\neg l_{n+j}/M\}$ and $/l/M \geq \min\{/\neg b/M, /k_i/M, /\neg k_{r+j}/M\}$ since $S \models_{L_2} \alpha, \beta$. Since either $/b/M$ or $/\neg b/M$ is $\mathbf{t}$, $/l/M \geq \min\{/l_i/M, /k_j/M, /\neg l_{n+l}/M, /\neg k_{r+k}/M\}$ and $/\gamma/M = \mathbf{t}$. Thus $S \models_{L_2} \gamma$.

The only-if part of (c) and (d) have been proved by (a) and (b) respectively.

(c): We need only consider l as an atom. Suppose $S \models_{L_2} l$. By the compactness theorem [84] of the classical logic, there exists a finite subset S' of S such that $S' \models_{L_2} a$. (Note that L_2 is identical with the classical logic.) Therefore, it suffices to consider only finite sets of ground program clauses. We prove the result by induction on the number of negative literals that appears in S .

Basis: Suppose that no negative literal appears in S . It is straightforward to show that for any ground atom l , $a \in T_S \uparrow^\omega$ iff $S \vdash_{Horn} l$, where T_S is the operator defined in Definition 4.12 (with Π being replaced by S). Suppose $S \models_{L_2} l$ for any $a \in B_L$. By Theorem 4.18, $a \in M_S$, where M_S is the least Herbrand model of S . By Theorem 4.20, $a \in T_S \uparrow^\omega$. Therefore, $S \vdash_{Horn} l$.

Induction: Assume that $S \models_{L_2} l$ implies $S \vdash_{Case} l$ when fewer than k negative literals appear in S , where $k > 0$. Suppose k negative literals appear in S and $S \models_{L_2} l$, there exists a clause $\delta \in S$ which has a negative literal $\neg b$ in its body. Let S' be the set obtained from S by replacing δ with δ' , which is obtained from δ with $\neg b$ deleted from it. Let S'' be the set obtained from S by replacing δ with $\delta'' : b$. Clearly, $S' \models_{L_2} l$ and $S'' \models_{L_2} l$ since $S' \models_{L_2} S$ and $S'' \models_{L_2} S$. Because S' and S'' have fewer than k negative literals, $S' \vdash_{Case} l$ and $S'' \vdash_{Case} l$ by inductive assumption. If δ' and δ'' are not involved in the derivation of l , then $S \vdash_{Case} l$. If both δ' and δ'' are involved, then $S \vdash_{Case} l \leftarrow \neg b$ and $S \vdash_{Case} l \leftarrow b$. By the reasoning-by-case rule, $S \vdash_{Case} l$.

(d): We construct a Herbrand interpretation I as follows. For each n -ary predicate p of L , the mapping p' assigned to p is the following: for $(t_1, \dots, t_n) \in U_L^n$,

$$p'(t_1, \dots, t_n) = \begin{cases} \mathbf{t} & \text{whenever } S \vdash_{Horn} p(t_1, \dots, t_n); \\ \mathbf{u} & \text{otherwise.} \end{cases}$$

It is straightforward to show that I is a Herbrand model of S . Thus, for any atom $a \in B_L$, if $S \not\models_{Horn} l$, then $/a/1 \neq t$. Therefore, $S \not\models_{L_3} l$. $\square$

The above lemma reveals that the semantics of the oriented derivations given in Definition 3.36 is captured by the semantics of different logics. Specifically, the semantics of $\vdash_{Horn}$ is captured by the semantics of L_3 and the semantics of $\vdash_{Case}$ is captured by the semantics of L_2 . Especially, $\vdash_{Horn}$ and $\vdash_{Case}$ are equivalent with $\models_{L_3}$ and $\models_{L_2}$ respectively when only normal programs are considered. We write $\vdash_{L_2}$ to denote $\vdash_{Case}$, and $\vdash_{L_3}$ to denote $\vdash_{Horn}$. It should be noted that Lemma 3.28 still hold when L_3 is replaced with any L_m , $m \geq 3$.

3.6.2 L_m -Saturated Answer Set Semantics for Extended Programs

In Section 3.2, all the concepts and definitions for the framework are introduced for any logic $\mathcal{L}$ although specific results are given for Lukasiewiczian many-valued first order logic L_m . In the definitions and results, the semantics of the logic $\mathcal{L}$ is represented by $\models_{\mathcal{L}}$. As indicated by the example $\Pi_1 : \{a \leftarrow \neg b; \neg a \leftarrow\}$, the use of $\models_{L_m}$ distorts the rule-like behavior of $\leftarrow$. $\models_{\mathcal{L}}$ ought to be replaced with an operator that preserves the semantics of the logic $\mathcal{L}$ and also the information embedded in the orientation of $\leftarrow$ as well. Clearly, $\vdash_{L_m}$ is a preferred candidate since it preserves the rule-like behavior of $\leftarrow$. Naturally for logic L_m , we can use $\vdash_{L_m}$ to replace $\models_{\mathcal{L}}$ in all the definitions in Section 3.2, and thus develop the framework based on $\vdash_{L_m}$.

Let Π be an extended program and $H \subseteq \Delta_{\Pi}$. The new framework is developed as follows. All the concepts defined in Section 3.2, 3.3 and 3.4 are redefined by replacing $\models_{\mathcal{L}}$ with $\vdash_{L_m}$. The redefined concepts include L_m -noncontradictory, positively L_m -founded, L_m -false-assumable, negatively L_m -founded, L_m -founded, L_m -admissible, L_m -saturated, L_m -preferable, L_m -stable, and L_m -well-founded hypothesis tuple, L_m -saturated closure, operator Ψ_{Π} , and ordinal power of Ψ_{Π} on H . Since only Definition 3.5, 3.6 and 3.7 (the definitions of noncontradictoriness, positive foundedness and false-assumability respectively) use $\models_{\mathcal{L}}$ directly and the rest of the concepts are based on the three definitions, we need only redefine the three concepts using $\vdash_{L_m}$.

Definition 3.37 Let Π be a program and $H = \langle P, N \rangle \subseteq \Delta_{\Pi}$. H is said to be L_m -noncontradictory of Π , if $H \uplus \Pi \not\models_{L_m} \neg l$ for each $l \in N$.

Definition 3.38 Let Π be a program and $H = \langle P, N \rangle \subseteq \Delta_{\Pi}$. A literal $l \in \delta_{\Pi}$ is said to be *positively L_m -founded* of Π wrt H if $H \uplus \Pi \vdash_{L_m} l$. H is also said to be

positively L_m -founded of Π if every ground literal $l \in P$ is positively L_m -founded of Π wrt H .

Definition 3.39 Let Π be a program and $H \subseteq \Delta_\Pi$. A ground literal $l \in \delta_\Pi$ is said to be L_m -false-assumable of Π wrt H if for every $H_l \subseteq \Delta_\Pi$ satisfying $H_l \uplus \Pi \vdash_{L_m} l$, there exists a literal $k \in H_l$ such that $H \uplus \Pi \vdash_{L_m} \neg k$.

By Lemma 3.28, $\vdash_{L_m}$ and $\models_{L_m}$ are equivalent when they are used in the expression for normal programs. Therefore, the redefined concepts and hence the framework are essentially the same as before when only normal programs are concerned.

Although $\vdash_{L_m}$ and $\models_{L_m}$ are not necessarily equivalent for extended programs, the framework newly developed with $\vdash_{L_m}$ has almost all the properties we presented in Section 3.2, 3.3 and 3.4 for the framework originally defined with $\models_{\mathcal{L}}$. The results are Lemma, Proposition or Theorem 3.4, 3.5, 3.6, 3.8, 3.9, 3.10, 3.11, 3.12, 3.19, 3.20, 3.21, 3.22, 3.23, 3.24 and 3.25, and 3.26.

Since only Proposition 3.4 and 3.5 directly use $\models_{L_m}$ and the rest of the results involve only the concepts we just redefined, we need only to revise the two propositions.

Proposition 3.29 Let Π be a program and $H \subseteq \Delta_\Pi$. Then for any S , $H \uplus \Pi \vdash_{L_m} S$ implies $H \cup \Pi \vdash_{L_m} S$.

Proof: The result follows immediately from the fact that for each $\alpha \in H \uplus \Pi$, $H \cup \Pi \vdash_{L_m} \alpha$. $\square$

Proposition 3.30 (Monotonicity) Let $H_1 \subseteq H_2 \subseteq \Delta_\Pi$ and l be a literal. Then $H_1 \uplus \Pi \vdash_{L_m} l$ implies $H_2 \uplus \Pi \vdash_{L_m} l$.

Proof: The result follows from that fact that for each ground program clause $\alpha : l \leftarrow l_1, \dots, l_n, \in H_1 \uplus \Pi$, there exists β such that $\beta \in H_2 \uplus \Pi$, where β is $l \leftarrow k_1, \dots, k_m$ and $\{k_1, \dots, k_m\} \subseteq \{l_1, \dots, l_n\}$. $\square$

All the results listed above still hold for the redefined concepts. For instance, $\uplus$ still has the monotonicity wrt L_m -foundedness as indicated by Proposition 3.30 and 3.8. The set AH_{L_m} of all L_m -admissible tuples and the set SH_{L_m} of all L_m -saturated tuple constitute complete semilattice as indicated by Theorem 3.20 and 3.26. The proofs of these results for the framework developed with $\vdash_{L_m}$ are essentially the same as or similar to the original proofs. They are given in the appendix (Section 3.8).

It should be noted that the list does not include Proposition 3.7. The proposition says that $H \uplus \Pi$ is consistent in the logic L_m if $H \subseteq \Delta_\Pi$ is L_m -noncontradictory of a program Π . Consider the following example.

Example 3.13 Consider the program Π [13, Example 4].

$$\begin{aligned} \Pi : \quad & a \leftarrow \text{not} p \\ & \neg a \leftarrow \text{not} q \\ & b \leftarrow \text{not} r \end{aligned}$$

Since p, q and r do not appear in the heads of any clauses, $H = \langle \emptyset, \{p, q, r\} \rangle$ is L_m -noncontradictory of Π . However, $H \uplus \Pi = \{a; \neg a; b\}$, which is not consistent in L_m . We also observe that $H \uplus \Pi \vdash_{L_m} a$ and $H \uplus \Pi \vdash_{L_m} \neg a$. a and $\neg a$ are a pair of complementary literals.

Definition 3.40 Let Π be an extended program in a language L and $H \subseteq \Delta_\Pi$. H is said to be L_m -sound if there does not exist $a \in B_L$ such that $H \uplus \Pi \vdash_{L_m} a$ and $H \uplus \Pi \vdash_{L_m} \neg a$.

An L_m -admissible hypothesis tuple may not be L_m -sound as indicated by Example 3.13. Let H be an L_m -saturated hypothesis tuple of an extended program Π . Then we use H_{sound} to denote the set of all hypothesis tuples I that are smaller than H , and are both L_m -admissible and L_m -sound of Π . I.e.

$$H_{\text{sound}} = \{I : I \subseteq H, \text{ and } I \text{ is } L_m\text{-admissible and } L_m\text{-sound.}\}$$

Since $\langle \emptyset, \emptyset \rangle$ is always L_m -admissible and L_m -sound, H_{sound} is not empty. (We consider only programs such that their logical forms are consistent in logic L_m .)

Definition 3.41 $I \subseteq \Delta_\Pi$ is said to be soundly L_m -saturated of Π if there exists an L_m -saturated hypothesis tuple H of Π such that I is a maximal (wrt $\subseteq$) member of H_{sound} . I is also said to be soundly L_m -preferable (L_m -well-founded, $\mathcal{L}$ -stable) if H is L_m -preferable (L_m -well-founded, $\mathcal{L}$ -stable).

Proposition 3.31 Let Π be a normal program and $H \subseteq \Delta_\Pi$. H is soundly L_m -saturated of Π iff H is L_m -saturated of Π .

Proof: It follows from Proposition 3.7 and Lemma 3.28 that an L_m -admissible hypothesis tuple of Π is L_m -sound of Π . The result follows immediately. $\square$

Although L_m -saturated hypothesis tuples of an extended program may not be L_m -sound, L_m -stable hypothesis tuples are always L_m -sound.

Proposition 3.32 Let Π be an extended program and $H \subseteq \Delta_\Pi$. H is soundly L_m -stable of Π iff H is L_m -stable of Π .

Proof: Let $H = \langle P, N \rangle$ be L_m -stable. By Proposition 3.12 and 3.11, H is also L_m -saturated. Then P contains all positively L_m -founded literals. Because P contains no pairs of complementary literals by Definition 3.12, H is L_m -sound. The result follows immediately. $\square$

Now we introduce various answer sets.

Definition 3.42 Let Π be a (disjunctive extended) program and $H = \langle S, N \rangle \subseteq \Delta_\Pi$. S is said to be an L_m -saturated answer set (L_m -preferable answer set, L_m -well-founded answer set, L_m -stable answer set) of Π if H is soundly L_m -saturated, (L_m -preferable, L_m -well-founded, L_m -stable) of Π .

Let $H = \langle S, N \rangle$ be soundly L_m -saturated of a program Π . Then an L_m -saturated answer set S is

$$S = \{l : H \uplus \pi \vdash_{L_m} l\},$$

where l is a literal and π is a semantical instance of Π wrt H .

Example 3.14 Consider the program Π in Example 3.13 again.

$$\begin{aligned} \Pi : \quad & a \leftarrow \text{not} p \\ & \neg a \leftarrow \text{not} q \\ & b \leftarrow \text{not} r \end{aligned}$$

Π has two soundly L_3 -saturated tuples $H_1 = \langle \{b, a\}, \{p\} \rangle$ and $H_2 = \langle \{b, \neg a\}, \{q\} \rangle$. Therefore, it has two L_3 -saturated sets $S_1 = \{b, a\}$ and $S_2 = \{b, \neg a\}$.

3.7 Syntactical Characterization of $\mathcal{L}$ -Saturated Semantics

In this section, we discuss the syntactical characteristics of an $\mathcal{L}$ -saturated hypothesis tuple.

In Definition 3.4, we introduced the notation $H \uplus \Pi$ for a program Π and a tuple $H = \langle P, N \rangle \subseteq \Delta_\Pi$. Clearly, such a notation is readily extended to an (infinite or finite) set of ground program clauses. I.e. for a set S of ground program clauses, $H \uplus S$ denotes the set obtained from S by deleting each $\text{not} l$ with $l \in N$. We need the following lemma to prove Proposition 3.34 and other propositions presented later.

Lemma 3.33 Let S be a set of extended program clauses in a language $\mathbf{L}$, $H = \langle P, N \rangle \subseteq \langle \text{Lit}_{\mathbf{L}}, \text{Lit}_{\mathbf{L}} \rangle$, $R \subseteq \text{Lit}_{\mathbf{L}}$ and $\gamma : l \leftarrow l_1, \dots, l_n, \text{not} k_1, \dots, \text{not} k_m$ be an extended program clause.

(a) If $H \uplus S \cup R \vdash_{L_m} \gamma$, then there exists

$\gamma' : l \leftarrow l_1, \dots, l_n, l'_1, \dots, l'_n, \text{not} k_1, \dots, \text{not} k_m, \text{not} k'_1, \dots, \text{not} k'_m$, such that $S \vdash_{L_m} \gamma'$, where for each $i \leq n'$, $l'_i \in R$ and each $j \leq m'$, $k'_j \in N$;

(b) If $S \vdash_{L_m} \gamma$, then there exists $\gamma'' : l \leftarrow l'_1, \dots, l'_n, \text{not} k'_1, \dots, \text{not} k'_m$, such that $H \uplus S \cup R \vdash_{L_m} \gamma''$, where for each $i \leq n'$, $l'_i \in \{l_1, \dots, l_n\}$ and $l'_j \notin R$ and for each $j \leq m'$, $k'_j \in \{k_1, \dots, k_m\}$ and $k'_j \notin N$.

Proof: It is straightforward to show the result by using structural induction based on Definition 3.36. $\square$

The following Proposition 3.34 reveals the syntactical characteristics of an L_m -false-assumable atom.

Proposition 3.34 *Let $H \subseteq \Delta_\Pi$ be L_m -noncontradictory of an extended program Π . A ground literal l is L_m -false-assumable of Π wrt H iff for any clause $\gamma : l \leftarrow l_1, \dots, l_n, \text{not} k_1, \dots, \text{not} k_m$, $n, m \geq 0$ such that $\Pi \vdash_{L_m} \gamma$, either there exists $i, 1 \leq i \leq n$, such that l_i is L_m -false-assumable wrt H , or there exists $j, 1 \leq j \leq m$, such that k_j is positively L_m -founded wrt H , i.e., $H \uplus \Pi \vdash_{L_m} k_j$.*

Proof: Let us call a literal l_i or k_j in the body of γ a witness if l_i is L_m -false-assumable wrt H or k_j is positively L_m -founded.

$\Rightarrow$ (Only-if): Suppose l is L_m -false-assumable wrt H . We show by structural induction that for any program clause γ if $\Pi \vdash_{L_m} \gamma$, then there is a witness of γ .

Basis: If $\gamma \in [\Pi]_L$ and there is no witness of γ , then l_i is not L_m -false-assumable and $H \uplus \Pi \not\vdash_{L_m} k_j$ for each $i \leq n$ and $j \leq m$. Then for each i , there exists a tuple $H_i = \langle P_i, N_i \rangle$ such that $H_{iN} \uplus \Pi \vdash_{L_m} l_i$, (where $H_{iN} = \langle \emptyset, N_i \rangle$) and $H \uplus \Pi \not\vdash_{L_m} k$ for each $k \in N_i$. Let $N' = \bigcup_{i=1}^n N_i \cup \{k_1, \dots, k_m\}$, and $H' = \langle \emptyset, N' \rangle$. Then $H' \uplus \Pi \vdash_{L_m} l_i$ by Proposition 3.30. Since $l \leftarrow l_1, \dots, l_n$ is in $H' \uplus \Pi$, $H' \uplus \Pi \vdash_{L_m} l$ by n applications of the Horn-type rule. However, $H \uplus \Pi \not\vdash_{L_m} k$ for each $k \in N'$. Thus, l is not L_m -false-assumable wrt H . Contradiction.

Induction: Let α and β be as in Definition 3.36 and γ is derived from α and β . Suppose that if the heads of α and/or β are L_m -false-assumable wrt H , then there exists a witness of the clauses.

For Horn-type rule: α is of the form $l \leftarrow k, l_1, \dots, l_n, \text{not} l_{n+1}, \dots, \text{not} l_{n+m}$ and β is of the form $k \leftarrow k_1, \dots, k_r, \text{not} k_{r+1}, \dots, \text{not} k_{r+s}$. If k is not L_m -false-assumable, the witness of α is a witness of γ . Otherwise, the witness of β is a witness of γ .

For proof-by-case rule: α is of the form $l \leftarrow b, l_1, \dots, l_n, \text{not} l_{n+1}, \dots, \text{not} l_{n+m}$ and β is of the form $i \leftarrow \neg b, k_1, \dots, k_r, \text{not} k_{r+1}, \dots, \text{not} k_{r+s}$. We need only

consider the case when b and $\neg b$ are the witness of α and β respectively. Since b and $\neg b$ may be the logical forms of **not** $\neg b$ and **not** b , there are four cases to consider (1) b and $\neg b$, (2) **not** $\neg b$ and **not** b , (3) **not** $\neg b$ and $\neg b$, (4) b and **not** b . In case (1) and (2), if there is no witness in γ , the similar proof in Basis will show contradiction. By Lemma 3.9, case (3) and (4) are eliminated. Therefore, both b and $\neg b$ can not be the witness, and γ has a witness.

Induction is completed.

$\Leftarrow$ (If): Suppose $\Pi \vdash_{L_m} \gamma$ implies the existence of a witness of γ . Let a tuple $H' = \langle P', N' \rangle \subseteq \Delta_\Pi$ satisfy $H' \uplus \Pi \vdash_{L_m} a$. By Lemma 3.33, there exists $\gamma' : l \leftarrow \text{not}n_1, \dots, \text{not}n_q$, where $\{n_1, \dots, n_q\} \subseteq N'$, such that $\Pi \vdash_{L_m} \gamma'$. Then there is a witness of γ' , i.e. there exists i such that $H \uplus \Pi \vdash_{L_m} n_i$. Therefore, l is L_m -false-assumable wrt H . $\square$

By Proposition 3.34, false-assumability also corresponds to underivability since for each oriented consequence with a false-assumable head, at least one literal in the body is to be false. Thus the idea of false-assumability also makes sense from a procedural point of view. It mirrors the spirit of *negation as failure* [7], which is considered as a procedural approximation of the closed world assumption.

The next proposition describes syntactical characteristics of a saturated tuple.

Proposition 3.35 *Let $H = \langle P, N \rangle \subseteq \Delta_\Pi$ be a hypothesis tuple of an extended program Π . H is L_m -saturated of Π if and only if the following holds.*

- (a) P and N are disjoint;
- (b) $l \in P$ iff there exists $\gamma : l \leftarrow \text{not}n_1, \dots, \text{not}n_q$, where $\{n_1, \dots, n_k\} \subseteq N$ such that $\Pi \vdash_{L_m} \gamma$;
- (c) $l \in N$ iff for any clause $\gamma : l \leftarrow l_1, \dots, l_n, \text{not}k_1, \dots, \text{not}k_m$, $n, m \geq 0$ such that $\Pi \vdash_{L_m} \gamma$, either there exists $i, 1 \leq i \leq n$, such that $l_i \in N$, or there exists $j, 1 \leq j \leq m$, such that $k_j \in P$.

Proof: Proof of Only-if: Suppose H is L_m -saturated. (a) follows from the L_m -noncontradictoriness and positive L_m -foundedness of H . (b) follows from Lemma 3.33 and Definition 3.10. (c) follows from Proposition 3.34 and Definition 3.10.

Proof of If: Suppose (a), (b) and (c) hold. It follows from Lemma 3.33 and (b) that H is positively L_m -founded. If there exists $l \in N$ such that $H \uplus \Pi \vdash_{L_m} l$, $l \in P$ follows from Lemma 3.33 and (b). This contradicts (a). Therefore, H is L_m -noncontradictory. For any $l \in N$ and any $H_l = \langle P_l, N_l \rangle$ such that $H_l \uplus \Pi \vdash_{L_m} l$, there exist $n_1, \dots, n_q \in N_l$ such that $\Pi \vdash_{L_m} l \leftarrow \text{not}n_1, \dots, \text{not}n_q$ by Lemma 3.33.

It follows from (b) and (c) that there exists j such that $H \uplus \Pi \vdash_{L_m} n_j$. Therefore, every $l \in N$ is L_m -false-assumable wrt H . It follows from Proposition 3.34 and Definition 3.10 that H is L_m -saturated. $\square$

Corollary 3.36 *Let $H = \langle P, N \rangle \subseteq \Delta_\Pi$ be a hypothesis tuple of an extended program Π . H is L_3 -saturated of Π if and only if the following holds.*

- (a) P and N are disjoint;
- (b) $l \in P$ iff there exists $\gamma: l \leftarrow \text{not}n_1, \dots, \text{not}n_q \in [\Pi]_{\mathbf{L}}$ such that $\{n_1, \dots, n_q\} \subseteq N$;
- (c) $l \in N$ iff for any clause $\gamma: l \leftarrow l_1, \dots, l_n, \text{not}k_1, \dots, \text{not}k_m \in [\Pi]_{\mathbf{L}}$, $n, m \geq 0$, either there exists $i, 1 \leq i \leq n$, such that $l_i \in N$, or there exists $j, 1 \leq j \leq m$, such that $k_j \in P$.

Proof: The result follows from Proposition 3.35 and Definition 3.36. $\square$

Lemma 3.33 indicates that the syntactical operator $\vdash_{L_m}$ is as strong as $\models_{L_m}$ when only normal programs are concerned. In general, the incorporation of hypotheses with a program Π based on $\uplus$ is weaker than the incorporation based on the set union $\cup$. However, the next proposition shows that for a normal program Π , an L_m -saturated scenario $H \uplus \Pi$ is as explanatory as a scenario $H \cup \Pi$ when only ground facts are concerned. According to the definition of correct answers, only ground literals are concerned in logic programming. Therefore, the L_m -saturated semantics of normal logic programs is as strong as the extension semantics for the hypothetical reasoning system. Furthermore, the L_m -saturated semantics captures the semantics of the underlying logic L_m . Note that in the expression $H \cup \Pi$ of the following proposition, Π is the logical form of the program Π , and $H = P \cup \{\neg a; a \in N\}$.

Proposition 3.37 *Let $H = \langle P, N \rangle \subseteq \Delta_\Pi$ be L_m -saturated of a normal program Π and $a \in \delta_\Pi$. Then $H \uplus \Pi \models_{L_m} a$ iff $H \cup \Pi \models_{L_m} a$.*

Proof: Suppose $H \uplus \Pi \models_{L_m} a$. $H \cup \Pi \models_{L_m} H \uplus \Pi$ by Proposition 3.4. Therefore, $H \cup \Pi \models_{L_m} a$.

Then N is the set of all atoms L_m -false-assumable wrt H and P is the set of all atoms a satisfying $H \uplus \Pi \models_{L_m} a$. Suppose $H \cup \Pi \models_{L_m} a$. Then $[H \cup \Pi]_{\mathbf{L}} \models_{L_m} a$ by Proposition 2.6. Since H is ground, $[H \cup \Pi]_{\mathbf{L}}$ is $H \cup [\Pi]_{\mathbf{L}}$ and $H \cup [\Pi]_{\mathbf{L}} \models_{L_m} a$. For any $\gamma: b \leftarrow l_1, \dots, l_n$ in $[\Pi]_{\mathbf{L}}$ with $b \in F$, there exists some i such that $\neg l_i \in H$ by Proposition 3.35. Let Π_1 be obtained from $[\Pi]_{\mathbf{L}}$ by deleting every clause γ with its head in N . Then $H \cup [\Pi]_{\mathbf{L}}$ is equivalent with $H \cup \Pi_1$, which is equivalent with

$H \cup (H \uplus \Pi_1)$. Thus $H \cup (H \uplus \Pi_1) \models_{L_m} a$. Since for any $b \in N$, b does not appear in $H \uplus \Pi_1$, $P \cup (H \uplus \Pi_1) \models_{L_m} a$. So $P \cup (H \uplus \Pi) \models_{L_m} a$ since $\Pi_1 \subseteq [\Pi]_L$. Note the fact that for any $b \in P$, $H \uplus \Pi \models_{L_m} b$. Therefore, $H \uplus \Pi \models_{L_m} a$. $\square$

In general, the equivalence in this proposition does not hold for L_m -admissible scenarios.

Example 3.15 Consider the following program:

$$\begin{array}{ll} \Pi_6 : & a \leftarrow b \\ & b \leftarrow a \\ & c \leftarrow \text{nota} \end{array}$$

Both a and b are L_m -false-assumable wrt $\emptyset$. $H = \langle \emptyset, \{b\} \rangle$ is L_m -admissible of Π_6 . $H \cup \Pi_6 \models_{L_m} c$ but $H \uplus \Pi_6 \not\models_{L_m} c$.

3.8 Appendix

We prove the following results for the framework developed with $\vdash_{L_m}$. The results are Lemma, Proposition or Theorem 3.6, 3.8, 3.9, 3.10, 3.11, 3.12, 3.19, 3.20, 3.21, 3.22, 3.23, 3.24 and 3.25, and 3.26.

Proposition 3.6 Let $H_1 \subseteq H_2 \subseteq \Delta_\Pi$. If H_2 is L_m -noncontradictory, So is H_1 .

Proof: The result follows immediately from Proposition 3.30. $\square$

Proposition 3.8 Let Π be a program and l be a ground literal. If $H_1 \subseteq H_2 \subseteq \Delta_\Pi$ and l is L_m -false-assumable of Π wrt H_1 , then l is L_m -false-assumable of Π wrt H_2 .

Proof: Because l is L_m -false-assumable of Π wrt H_1 , for any H_l satisfying $H_l \uplus \Pi \vdash_{L_m} l$, there exists $k \in H_l$ such that $H_l \uplus \Pi \vdash_{L_m} \neg k$. By Proposition 3.30, $H_2 \uplus \Pi \vdash_{L_m} \neg k$. Hence, l is L_m -false-assumable of Π wrt H_2 . $\square$

Lemma 3.9 Let $H \subseteq \Delta_\Pi$ be L_m -noncontradictory of a program Π . For $l \in \delta_\Pi$, $H \uplus \Pi \not\models_{L_m} l$ if l is L_m -false-assumable of Π wrt H .

Proof: Suppose l is L_m -false-assumable of Π wrt H and $H \uplus \Pi \vdash_{L_m} l$. Then $H_N \uplus \Pi \vdash_{L_m} l$ where $H_N = \langle \emptyset, N \rangle$. Thus there exists $k \in N$ such that $H \uplus \Pi \vdash_{L_m} \neg k$. This contradicts the fact that H is L_m -noncontradictory. $\square$

Proposition 3.10 Let $H = \langle P, N \rangle \subseteq \Delta_\Pi$ be L_m -admissible of a program Π , T be a set of positively L_m -founded literals of Π wrt H , and F be a set of L_m -false-assumable literals of Π wrt H . Then $H' = H \cup \langle T, F \rangle$ is L_m -admissible of Π .

Proof: Let $H' = \langle P', N' \rangle$. By Proposition 3.30, H' is positively L_m -founded. Since for each $l \in N'$, l is L_m -false-assumable wrt H , l is also L_m -false-assumable wrt H' by Proposition 3.8. Therefore, H' is negatively L_m -founded.

We then prove H' is L_m -noncontradictory. Suppose there exists $l \in N'$ such that $H' \uplus \Pi \vdash_{L_m} \neg l$. Then $H'_N \uplus \Pi \vdash_{L_m} \neg l$ by Definition 3.4, where $H'_N = \langle \emptyset, N' \rangle$.

Since $\neg l$ is L_m -false-assumable wrt H , there exists $k \in N'$ such that $H \uplus \Pi \vdash_{L_m} k$. This contradicts Lemma 3.9 since k is L_m -false-assumable wrt H . Therefore, H' is L_m -noncontradictory. By Definition 3.8 and 3.9, H' is L_m -admissible. $\square$

The proofs of Proposition 3.11, 3.12 are the same as before.

Lemma 3.19 Let DS be a nonempty directed subset of AH_{L_m} . Then $UH = \bigcup_{H \in DS} H$ is L_m -admissible of Π .

Proof: We first prove the following fact:

For any ground literal l , $UH \uplus \Pi \vdash_{L_m} l$ iff there exists an element $H \in DS$ such that $H \uplus \Pi \vdash_{L_m} l$.

One direction is trivial by Proposition 3.30. Suppose $UH \uplus \Pi \vdash_{L_m} l$. By the definition of $\vdash_{L_m}$, there exists a finite subset FSP of $UH \uplus \Pi$ such that $FSP \vdash_{L_m} l$. Then there exists a finite subset FS of UH such that $UFS \uplus \Pi \vdash_{L_m} l$, where $UFS \cup_{H \in FS} H$. Because DS is directed and FS is a finite subset of DS , there exists an upper bound H' of FS in DS . Clearly $UFS \subseteq H'$. By Proposition 3.5, $H' \uplus \Pi \vdash_{L_m} l$.

We then prove that UH is L_m -noncontradictory. If it is not, there exists $l \in UH$ such that $UH \uplus \Pi \vdash_{L_m} \neg l$ and there exists an element $H_l \in DS$ such that $l \in H_l$. By the above fact, there exists some $H \in DS$ such that $H \uplus \Pi \vdash_{L_m} \neg l$. Let H' be the least upper bound of $\{H, H_l\}$. By Proposition 3.30, $H' \uplus \Pi \vdash_{L_m} \neg l$. Since $l \in H'$, H' is not L_m -noncontradictory. This contradicts $H' \in AH_{L_m}$.

We then prove UH is L_m -founded. Let $UH = \langle UP, UN \rangle$. For each $l \in UP$, there exists some $H_l = \langle P_l, N_l \rangle \in DS$ such that $l \in P_l$. Then $H_l \uplus \Pi \vdash_{L_m} l$. It follows from Proposition 3.30 that $UH \uplus \Pi \vdash_{L_m} l$. So UH is positively L_m -founded. For each $l \in UN$, there exists some $H_l = \langle P_l, N_l \rangle \in DS$ such that $l \in N_l$. l is L_m -false-assumable wrt H_l . By Proposition 3.8, l is L_m -false-assumable wrt UH since $H_l \subseteq UH$. Therefore, UH is L_m -admissible. $\square$

Theorem 3.20

- (a) $(AH_{L_m}, \subseteq)$ is a complete semilattice;
- (b) For every L_m -admissible tuple $H \in AH_{L_m}$, there exists at least one L_m -preferable tuple $H_p \in AH_{L_m}$ such that $H \subseteq H_p$.

Proof: Proof of (a): Let SS be a nonempty subset of AH_{L_m} , and SLB be the set of lower bounds of SS , i.e. $SLB = \{H : H \subseteq \cap_{K \in SS} K, \text{ and } H \in AH_{L_m}\}$. We prove that $GLB = \cup_{H \in SLB} H$ is the greatest lower bound of SS . Clearly, $GLB \subseteq K$ for each $K \in SS$, and $H \subseteq GLB$ for each $H \in SLB$. Then we need only to show that $GLB \in AH_{L_m}$. Let $GLB = \langle GP, GN \rangle$. We show GLB is L_m -admissible of Π .

Since there exists $K \in SS$ such that $GLB \subseteq K$ and K is L_m -noncontradictory, it follows from Proposition 3.6 that GLB is L_m -noncontradictory. We now prove that GLB is L_m -founded. For each literal $l \in GP$, there exists some $H_l = \langle P_l, B_l \rangle \in SLB$ such that $l \in P_l$ and $H_l \uplus \Pi \vdash_{L_m} l$. By Proposition 3.30, $GLB \uplus \Pi \vdash_{L_m} l$ since $H_l \subseteq GLB$. Therefore, GLB is positively L_m -founded. For each literal $l \in GN$, there exists some $H_l = \langle P_l, B_l \rangle \in SLB$ such that $l \in N_l$ and l is L_m -false-assumable wrt H_l . By Proposition 3.8, l is L_m -false-assumable wrt GLB since $H_l \subseteq GLB$. Therefore, GLB is negatively L_m -founded. GLB is L_m -admissible and is indeed the greatest lower bound of SS .

Let DS be a nonempty directed subset of AH_{L_m} , and $LUB = \cup_{H \in DS} H$. Then LUB is a least upper bound of DS since LUB is L_m -admissible of Π by Lemma 3.19, i.e. $LUB \in AH_{L_m}$.

Proof of (b): For any element $H \in AH_{L_m}$, let $UH = \{H' \in AH_{L_m} : H \subseteq H'\}$. Since $H \in UH$, UH is not empty. We claim that every totally ordered subset S of UH has an upper bound in UH . Let $US = \cup_{H' \in S} H'$. By Lemma 3.19, US is L_m -admissible. Clearly, US is an upper bound of S and $US \in UH$. Since UH is a nonempty partial ordered set in which every totally ordered subset has an upper bound, by Proposition 3.18 (Zorn's Lemma), there exists a maximal element PH in

UH . By the definition of UH , $H \subseteq PH$ and PH is also a maximal element of AH_{L_m} .
□

Proposition 3.21 Ψ_Π is monotonic when $\mathcal{L}$ is L_m .

Proof: The result follows directly from Proposition 3.30 and Proposition 3.8. □

The proofs of Proposition 3.22, 3.23, 3.24 and 3.25 are the same as before.

Theorem 3.26 Let SH_{L_m} be the set of all L_m -saturated hypothesis tuples of a program Π .

- (a) $(SH_{L_m}, \subseteq)$ is a complete semilattice;
- (b) H_{L_m-wf} is the least element of $(SH_{L_m}, \subseteq)$.

Proof:

Proof of (a): Let SS be a nonempty subset of SH_{L_m} . Let SLB be the set of lower bounds of SS , i.e. $SLB = \{H : H \subseteq \cap_{K \in SS} K, \text{ and } H \in SH_{L_m}\}$. Let $GLB = \langle GP, GN \rangle = \cup_{H \in SLB} H$. We first prove that GLB is L_m -admissible.

For any $K \in SS$, $GLB \subseteq K$. It follows from Proposition 3.6 that GLB is L_m -noncontradictory because K is. For each literal $l \in GP$, there exists some $H_l = \langle P_l, N_l \rangle \in SLB$ such that $l \in P_l$ and $H_l \uplus \Pi \vdash_{L_m} l$. By Proposition 3.30, $GLB \uplus \Pi \vdash_{L_m} l$ since $H_l \subseteq GLB$. Therefore, GLB is positively L_m -founded. For each literal $l \in GN$, there exists some $H_l = \langle P_l, N_l \rangle \in SLB$ such that $l \in N_l$ and l is L_m -false-assumable wrt H_l . By Proposition 3.8, l is L_m -false-assumable wrt GLB since $H_l \subseteq GLB$. Therefore, GLB is negatively L_m -founded. Thus we proved that GLB is L_m -admissible. By Proposition 3.24, the L_m -saturated closure GLB^* of GLB is GLB_{fix} . Because $GLB \subseteq GLB_{fix}$, GLB^* is an upper bound of SLB by Proposition 3.21. It follows from Definition 3.27 that GLB^* is a lower bound of SS since for every $K \in SS$, $GLB \subseteq K$ and K is L_m -saturated.

Let DS be a nonempty directed subset of SH_{L_m} , and $LUB = \cup_{H \in DS} H$. By Lemma 3.19, LUB is L_m -admissible. Then by Proposition 3.24, the L_m -saturated closure LUB^* of LUB is LUB_{fix} . LUB^* is an upper bound of DS because $LUB \subseteq LUB^*$. For any upper bound $UB \in SH_{L_m}$ of DS , $LUB \subseteq UB$. Because Ψ_Π is monotonic by Proposition 3.21, $LUB_{fix} \subseteq UB_{fix}$. Because UB is L_m -saturated, $UB = UB_{fix}$. Hence, $LUB^* \subseteq UB$ and LUB^* is the least upper bound of DS .

Proof of (b): By Proposition 3.22, every L_m -saturated hypothesis tuple is a fixed point of the operator Ψ_Π . Since H_{L_m-wf} is the least fixed point of Ψ_Π by Proposition 3.25, it is the least element of SH_{L_m} . $\square$

Chapter 4

Further Analyses: Normal Logic Program

In this chapter, we shall show that almost all the existing semantics proposed for (disjunctive) normal logic programs fit into our framework by proving that each of them is equivalent to a semantics of the framework defined in Section 4.1. As a consequence of the equivalences, we shall also show that the L_2 -preferable semantics is a new semantics for both normal and disjunctive normal programs, and the L_2 -well-founded and the L_2 -stable semantics are new semantics for disjunctive programs. We shall also show how another logic — autoepistemic logic is used in our framework.

4.1 Partial Model Representation of $\mathcal{L}$ -Saturated Semantics

In Section 2.3.2, we discussed partial model semantics. We now present the partial model equivalence of the $\mathcal{L}$ -saturated semantics. First, we prove a property of an L_m -admissible tuple of a normal program.

Lemma 4.1 *Let $H \subseteq \Delta_\Pi$ be L_m -noncontradictory of a normal program Π and F be the set of all atoms L_m -false-assumable wrt H . Then $I = \langle T, F \rangle$, where $T = B_L - F$, is a 2-valued Herbrand model for Π .*

Proof: Let $H = \langle P, N \rangle$. By Proposition 2.4, it suffices to show that I is a Herbrand model for $[\Pi]_L$. Let $\gamma : a \leftarrow a_1, \dots, a_n, \text{not} b_1, \dots, \text{not} b_m$ be a clause in $[\Pi]_L$. If $\gamma/I \neq \mathbf{t}$, i.e. $\gamma/I = \mathbf{f}$, then $a \in F$, $\{a_1, \dots, a_n\} \subseteq T$ and $\{b_1, \dots, b_m\} \subseteq F$. Since $a \in F$ is L_m -false-assumable, there exists i such that $H \uplus \Pi \vdash_{L_m} b_i$ by Proposition 3.34. And by Lemma 3.33, there exists $\gamma' : b_i \leftarrow \text{not} c_1, \dots, \text{not} c_k$ such that $\Pi \vdash_{L_m} \gamma'$.

where $\{c_1, \dots, c_k\} \subseteq N$. Since $b_i \in F$ is L_m -false-assumable, by Proposition 3.34 again, there exists j such that $H \uplus \Pi \vdash_{L_m} c_j$. Thus, H is not L_m -noncontradictory. Contradiction. $\square$

Proposition 4.2 *Every L_m -admissible tuple of a normal program Π is a partial model of Π .*

Proof: Let $H = \langle P, N \rangle$ be L_m -admissible, and F be the set of all atoms L_m -false-assumable wrt H . Then $N \subseteq F$ because H is negatively L_m -founded. Let $T = B_L - F$. For each $a \in P$, $H \uplus \Pi \vdash_{L_m} a$. Thus $a \notin F$ by Lemma 3.9. So $P \subseteq T$. Therefore, $\langle T, F \rangle$ is an extension of I . By Lemma 4.1, $\langle T, F \rangle$ is a model of Π . Hence, H is a partial model of Π . $\square$

Because of the above proposition, an L_m -admissible tuple may well be called an L_m -admissible model of a program. The same argument applies to L_m -saturated, L_m -preferable, L_m -stable and L_m -well-founded tuples too. It is worthy of note that an L_m -admissible model $H = \langle T, F \rangle$ is always positively L_m -founded. Then for each $a \in T$, $H \uplus \Pi \vdash_{L_m} a$. T amounts to all the positive hypotheses in H , which are also explicable by H and Π while F amounts to all the negative hypotheses in H . The partial model semantics may be defined correspondingly. I.e. the L_m -saturated ($\mathcal{L}$ -preferable, L_m -stable or L_m -well-founded) semantics is defined as the set of all L_m -saturated (L_m -preferable, L_m -stable or L_m -well-founded) models.

Example 4.1 Consider the following program Π_1 and Π_5 in Example 3.9:

$\Pi_1 :$	$a \leftarrow \text{not} b$	$\Pi_5 :$	$a \leftarrow \text{not} b$
	$b \leftarrow \text{not} a$		$b \leftarrow \text{not} a$
	$d \leftarrow a$		$d \leftarrow \text{not} a$
	$d \leftarrow b$		$d \leftarrow \text{not} b$

Both Π_1 and Π_5 have three L_3 -admissible models. They are $M_1 = \langle \{d, a\}, \{b\} \rangle$, $M_2 = \langle \{d, b\}, \{a\} \rangle$ and $M_3 = \langle \emptyset, \emptyset \rangle$. Among them, M_1 and M_2 are L_3 -preferable models. $M_4 = \langle \{d\}, \emptyset \rangle$ is also an L_2 -admissible model of Π_1 , but not of Π_5 . M_1 and M_2 are still L_2 -preferable models of Π_1 and Π_5 and M_3 is still an L_2 -admissible model of Π_5 .

It was explicitly pointed out in [63] that models that can serve as the intended models of a program such as the stable model, well-founded model, etc. have the following properties. Let M be such a semantical model.

P1: If $a \leftarrow l_1, \dots, l_n \in [\Pi]_{\mathbf{L}}$ and $M \models l_1, \dots, l_n$, then $M \models a$.

P2: If $M \models a$, then there exists $a \leftarrow l_1, \dots, l_n \in [\Pi]_{\mathbf{L}}$ such that $M \models l_1, \dots, l_n$.

P3: If for every clause $a \leftarrow l_1, \dots, l_n \in [\Pi]_{\mathbf{L}}$ with head a , there exists i such that $M \models \neg l_i$, then $M \models \neg a$.

P4: If $M \models \neg a$, then for every clause $a \leftarrow l_1, \dots, l_n \in [\Pi]_{\mathbf{L}}$ with head a , there exists i such that $M \models \neg l_i$.

It follows from Corollary 3.36 that an L_3 -saturated model has the properties P1-P4. Let $\Pi^* = \{\alpha : \Pi \vdash_{L_2} \alpha\}$. (I.e., Π^* is the set of all oriented consequences of Π). Let P1'-P4' be the properties obtained from P1-P4 by replacing $[\Pi]_{\mathbf{L}}$ with Π^* . It also follows from Proposition 3.35 that an L_2 -saturated model has the properties P1'-P4'. This observation suggests that L_m -saturated models are proper candidates of intended models of a program. Indeed, we have defined various semantics using L_m -saturated models.

4.2 Preferential Extension Semantics

Based on Poole's framework, Dung proposed the *preferential extension semantics* for normal logic programs [12]. This section establishes the relationship of the $\mathcal{L}$ -preferable semantics and the *preferential extension semantics*. We will show that the latter is a special case of the former by proving that the latter coincides with L_3 -preferable semantics. Since it has been shown that the preferential extension semantics is equivalent to the regular model semantics proposed by You and Yuan [93], we also show that the regular model semantics fits into the framework.

Poole revised his framework as a triple $(\mathcal{F}, \Delta, C)$, where C is a set of sentences called the set of *constraints* [64]. Constraints are sentences with which scenarios must be consistent. Then a *scenario* of $(\mathcal{F}, \Delta, C)$ is a set $H \cup \mathcal{F}$ where H is a set of ground instances of elements of Δ such that $H \cup \mathcal{F} \cup C$ is consistent. ([64] Definition 7.4) The definitions of explicability and extensions are correspondingly changed.

We now proceed to present the preferential extension semantics. Let Π be a normal program in a first order language $\mathbf{L}$. For each predicate symbol p of the language $\mathbf{L}$, a new predicate symbol $\bar{p}$ of the same arity is introduced to $\mathbf{L}$. All these new predicates are called *abductable* predicates. Atoms of abductable predicates are called *abductable* atoms. With the introduction of these abductable predicates, a reasoning system $(\bar{\Pi}, \bar{\Delta}_{\mathbf{L}}, C_{\mathbf{L}})$ in the enlarged language $\mathbf{L}$ can be defined as follows. $\bar{\Pi}$, called *abductive program* of Π , is the program obtained from Π by replacing every negative literal $\text{not}p(\vec{x})$ with the corresponding abductable atom $\bar{p}(\vec{x})$, the set $\bar{\Delta}_{\mathbf{L}}$ of all possible

hypotheses is the set of all abductable atoms and the set C_L of constraints consists of $\{\forall(\leftarrow p(\vec{x}) \wedge \bar{p}(\vec{x}))\}$ for every predicate p of L . Therefore, in such a hypothetical reasoning system, each hypothesis is a ground abductable atom. Since only ground instances of elements in $\bar{\Delta}_L$ are hypotheses, we also use $\bar{\Delta}_L$ to denote the set of all ground instances of abductable atoms. From context, no confusion can be caused.

The basic idea of the semantics is the notion of acceptability of a hypothesis with respect to a scenario. The *contrary* to a hypothesis $\bar{p}(\vec{a})$ is defined as the corresponding ordinary ground atom $p(\vec{a})$, and a set $\bar{H}_{p(\vec{a})} \subseteq \bar{\Delta}_L$ is called an *evidence* of an ordinary ground atom $p(\vec{a})$ if $\bar{\Pi} \cup \bar{H}_{p(\vec{a})} \models_{\mathcal{CL}} p(\vec{a})$, where $\mathcal{CL}$ stands for the classical logic. For a scenario $S = \bar{\Pi} \cup \bar{H}$ of $(\bar{\Pi}, \bar{\Delta}_L, C_L)$, where $\bar{H} \subseteq \bar{\Delta}_L$, let $inout(S) = \{a : S \models_{\mathcal{CL}} a\}$ where a is a ground (ordinary or abductable) atom. Clearly, $\bar{H} \subseteq inout(S)$. $\bar{H}$ is considered as the input to the reasoning system, and the rest of $inout(S)$ is considered as the output of the system for the input $\bar{H}$. A hypothesis $\bar{a}$ is *acceptable* wrt a scenario S if for every evidence $\bar{H}_a$ of the contrary a to $\bar{a}$, $\bar{H}_a \cup inout(S) \cup C_L$ is inconsistent. (This is equivalent to that there exists at least one hypothesis $\bar{b} \in \bar{H}_a$ such that $S \models_{\mathcal{CL}} b$ where b is the contrary to $\bar{b}$.) A scenario is *admissible* if every hypothesis in the scenario is acceptable with respect to the scenario, and a *preferential extension* of an abductive program is a maximal (wrt set inclusion) admissible scenario of the program.

In the following lemmas and theorem, Π is a program in a language L , and $\bar{\Pi}$ is the abductive program of Π . $H = \langle P, N \rangle \subseteq \Delta_\Pi$ and $\bar{H} \subseteq \bar{\Delta}_L$ satisfy the equation $\bar{H} = \{\bar{a} : a \in N\}$, where $\bar{a}$ is the corresponding abductable atom of the ordinary atom a .

Theorem 4.3

- (a) $H \uplus \Pi$ is an L_3 -admissible scenario of (Π, Δ_L) iff $\bar{\Pi} \cup \bar{H}$ is admissible scenario of $(\bar{\Pi}, \bar{\Delta}_L, C_L)$;
- (b) $H \uplus \Pi$ is an L_3 -preferable scenario of (Π, Δ_L) iff $\bar{\Pi} \cup \bar{H}$ is a preferential extension of $\bar{\Pi}$.

Proof: The result follows directly from Lemma 4.5 and Lemma 4.6. $\square$

By Theorem 4.3, the preferential extension semantics coincides with the L_3 -preferable semantics.

We now prove the lemma.

Lemma 4.4 *Let a be a ground ordinary atom of L . $H \uplus \Pi \vdash_{L_3} a$ iff $\bar{\Pi} \cup \bar{H} \models_{\mathcal{CL}} a$.*

Proof:

$H \uplus \Pi \vdash_{L_3} a$;

iff there exist $\{a_1, \dots, a_n\} \subseteq N$ such that $\Pi \vdash_{L_3} a \leftarrow \text{nota}_1, \dots, \text{nota}_n$ by Lemma 3.33;

iff there exist $\{\bar{a}_1, \dots, \bar{a}_n\} \subseteq \bar{H}$ such that $\bar{\Pi} \vdash_{L_3} a \leftarrow \bar{a}_1, \dots, \bar{a}_n$ (since abductable atoms never appear in the heads of clauses);

iff $\bar{\Pi} \cup \bar{H} \vdash_{L_3} a$ by n applications of the Horn-type rule;

iff $\bar{\Pi} \cup \bar{H} \vdash_{L_2} a$ because $\bar{\Pi} \cup \bar{H}$ does not contain negations;

iff $\bar{\Pi} \cup \bar{H} \models_{L_2} a$ by Lemma 3.28;

iff $\bar{\Pi} \cup \bar{H} \models_{\mathcal{CL}} a$ because L_2 is the same as $\mathcal{CL}$.

□

Lemma 4.5 H is L_3 -noncontradictory of Π iff $\bar{\Pi} \cup \bar{H}$ is a scenario.

Proof:

$\Leftarrow$ (If): Suppose H is not L_3 -noncontradictory of Π . There exists $a \in N$ such that $H \uplus \Pi \vdash_{L_3} a$. Then there exists $\bar{a} \in \bar{H}$ such that $\bar{\Pi} \cup \bar{H} \models_{\mathcal{CL}} a$ by Lemma 4.4. Since $\bar{a} \in \bar{H}$, none of the models for $\bar{\Pi} \cup \bar{H}$ is a model for $\leftarrow a \wedge \bar{a}$. Thus $\bar{\Pi} \cup \bar{H} \cup C_L$ is not consistent, and $\bar{\Pi} \cup \bar{H}$ is not a scenario.

$\Rightarrow$ (Only-if): Suppose H is L_3 -noncontradictory of Π . Let $M = \{a : \bar{\Pi} \cup \bar{H} \models_{\mathcal{CL}} a\}$, where a is a ground (ordinary or abductable) atom. Then M is the least Herbrand model of $\bar{\Pi} \cup \bar{H}$ by Theorem 4.18. Because $H \uplus \Pi \not\vdash_{L_3} a$ for each $a \in N$, $\bar{\Pi} \cup \bar{H} \not\models_{\mathcal{CL}} a$ for each $\bar{a} \in \bar{H}$. Thus for any ground atom a , either $a \notin M$ or $\bar{a} \notin M$. M is also a model for C_L . Therefore, $\bar{\Pi} \cup \bar{H} \cup C_L$ is consistent and $\bar{\Pi} \cup \bar{H}$ is a scenario. □

Lemma 4.6 Let a be a ground ordinary atom of L and $S = \bar{\Pi} \cup \bar{H}$ be a scenario. a is L_3 -false-assumable of Π wrt H iff the hypothesis $\bar{a}$ is acceptable wrt the scenario S .

Proof:

a is L_3 -false-assumable wrt H ;

iff for each $H_a = \langle P_a, N_a \rangle \subseteq \Delta_L$ satisfying $H_a \uplus \Pi \vdash_{L_3} a$, there exists $b \in N_a$ such that $H \uplus \Pi \vdash_{L_3} b$;

iff for each $\bar{H}_a \subseteq \bar{\Delta}_L$ satisfying $\bar{\Pi} \cup \bar{H}_a \models_{L_3} a$, there exists $\bar{b} \in \bar{H}_a$ such that $\bar{\Pi} \cup \bar{H} \models_{L_3} b$ by Lemma 4.4;

iff $\bar{a}$ is acceptable wrt S .

(Note that in the above proof, $\bar{H}_a = \{\bar{b} : b \in N_a\}$.) □

4.3 The 3-valued Stable Model Semantics

Przymusiński proposed *3-valued stable model semantics*, which extended the (2-valued) stable model semantics, which will be introduced in Section 4.5, such that the semantics is defined for the class of all (disjunctive) normal programs. We show that the semantics is equivalent to the $\mathcal{L}$ -saturated semantics.

Definition 4.1 Let Π be a program in a 3-valued language $\mathbf{L}$ and I be a Herbrand interpretation of $\mathbf{L}$. The *extended GL-transformation of Π modulo I* , denoted as $\frac{\Pi}{I}$, is the set obtained from the Herbrand instantiation $[\Pi]_{\mathbf{L}}$ of Π by replacing every **nota** appearing in $[\Pi]_{\mathbf{L}}$ with the truth value $/\neg a/I$ denoted by $\neg a$ in the Herbrand interpretation I , i.e. **nota** is replaced with **t**, **u** and **f** respectively when $/a/I$ is **f**, **u** and **t**.

We assume all **t**'s have been removed from $\frac{\Pi}{I}$ since **t** does not play any role in the body of a clause.

Every Herbrand interpretation of a 3-valued language can be identified with a consistent partial interpretation. This identification is made throughout.

Proposition 4.7 Let Π be a normal program in a 3-valued language $\mathbf{L}$ and I be a Herbrand interpretation of $\mathbf{L}$. The extended GL-transformation $\frac{\Pi}{I}$ of Π modulo I has the least Herbrand model.

Proof: We prove the result by constructing the least Herbrand model of $\frac{\Pi}{I}$. In the following proof, all interpretations are Herbrand interpretations. Let Π_1 be obtained from $\frac{\Pi}{I}$ by deleting each sentence which contains an **f**. Clearly, for any interpretation M of $\mathbf{L}$, M is a model of Π_1 iff M is a model of $\frac{\Pi}{I}$. Π_1 does not contain **t** or **f**. Let Π_2 consist of all and only sentences in Π_1 that do not contain **u**. Let $T \subseteq B_{\mathbf{L}}$ be the set of all and only ground atoms that are logical consequences of Π_2 . By Theorem 4.20, $T = \text{Ifp}(T_{\Pi_2})$. Thus,

Fact 1: For any atom $a \in B_{\mathbf{L}}$, $a \in T$ iff there exists $\alpha : a \leftarrow a_1, \dots, a_n$ such that $\alpha \in \Pi_2$ and $\{a_1, \dots, a_n\} \subseteq T$.

Since Π_1 is a superset of Π_2 , every atom in T is also a logical consequence of Π_1 . Furthermore, $\langle T, \emptyset \rangle$ is also a Herbrand model of Π_1 by Fact 1 and the fact that each sentence in $\Pi_1 - \Pi_2$ contains a **u**. Therefore, T contains all ground atoms that are logical consequences of Π_1 . Let Π_3 be obtained from Π_1 by deleting from the bodies of Π_1 all atoms in T . By Fact 1, for any atom $a \in B_{\mathbf{L}}$, $a \in T$ iff $a \in \Pi_3$, and for any interpretation M of $\mathbf{L}$, M is a model of Π_1 iff M is a model of Π_3 . Let U be the set of ground atoms for which there exists $\beta : a \leftarrow u, \dots, u$ such that $\beta \in \Pi_3$. Then,

$M = \langle T, F \rangle$ is a Herbrand model of Π_3 , where $F = B_L - (T \cup U)$. Furthermore, M is the least Herbrand model of Π_3 . Hence, M is the least Herbrand model of Π_1 , and thus, of $\frac{\Pi}{I}$. $\square$

Definition 4.2 Let Π be a normal program in a 3-valued language L and I be a Herbrand interpretation of L . I is called a 3-valued stable model if I is the least Herbrand model of the extended GL-transformation $\frac{\Pi}{I}$ of Π modulo I .

We now introduce 3-valued stable-by-case model using $\vdash_{L_2}$.

Definition 4.3 Let Π be a program and $\Pi^* = \{\alpha : \Pi \vdash_{L_2} \alpha\}$. A 3-valued stable-by-case model is defined as a 3-valued stable model of Π^* .

The following proposition shows why the 3-valued interpretation in Definition 4.2 is called a 3-valued stable "model".

Proposition 4.8 [73] Let Π be a program in a 3-valued language L . Any 3-valued stable model of a program Π is a minimal Herbrand model of Π .

We need the following lemma to prove the main result of this section Theorem 4.10.

Lemma 4.9 Let Π be a normal program, $M = \langle T, F \rangle$ be a consistent partial interpretation and a a ground atom. Then

$$(a) \ M \uplus \Pi \vdash_{L_3} a \text{ iff } \frac{\Pi}{M} \vdash_{L_3} a;$$

$$(b) \ M \uplus \Pi \vdash_{L_2} a \text{ iff } \frac{\Pi^*}{M} \vdash_{L_3} a;$$

(where Π^* is the set of all oriented consequences, $\frac{\Pi}{M}$ and $\frac{\Pi^*}{M}$ are the extended GL-transformation of Π and Π^* respectively, as defined in Definition 4.1.)

Proof: Proof of (a):

It follows from the fact that $M \uplus \Pi$ and $\frac{\Pi}{M}$ have the same set of clauses that do not contain $\neg$, **u** or **f**.

Proof of (b):

Suppose $M \uplus \Pi \vdash_{L_2} a$. By Lemma 3.33, there exists α such that $\Pi \vdash_{L_2} \alpha$, where α is $a \leftarrow \text{not}b_1, \dots, \text{not}b_n$ and $\{b_1, \dots, b_n\} \subseteq F$. Then $\alpha \in \Pi^*$. Therefore, $\frac{\Pi^*}{M} \vdash_{L_3} a$. Suppose $\frac{\Pi^*}{M} \vdash_{L_3} a$. By Lemma 3.33 and Definition 4.1, there exists α such that $\Pi^* \vdash_{L_3} \alpha$, where α is $a \leftarrow \text{not}b_1, \dots, \text{not}b_n$ and $\{b_1, \dots, b_n\} \subseteq F$. Since Π^* is closed under $\vdash_{L_3}$, $\alpha \in \Pi^*$ and $\Pi \vdash_{L_2} \alpha$. By Lemma 3.33, $M \uplus \Pi \vdash_{L_2} a$. $\square$

Now we come to the major result of this section. The next theorem shows that the $\mathcal{L}$ -saturated models correspond to different 3-valued stable models by choosing the logic $\mathcal{L}$ as different Lukasiewiczian many-valued logics.

Theorem 4.10 *Let Π be a normal program and $M = \langle T, F \rangle$ a consistent partial interpretation.*

- (a) *M is L_3 -saturated iff M is a 3-valued stable model of Π .*
- (b) *M is L_2 -saturated iff M is a 3-valued stable-by-case model of Π .*

Proof: Proof of (a):

$\Leftarrow$ (If). Suppose M is a 3-valued stable model of Π . Then M is the least Herbrand model of $\frac{\Pi}{M}$. Since the least Herbrand model is unique, M is the model constructed in the proof of Proposition 4.7 as the least Herbrand model of $\frac{\Pi}{M}$. Then, M satisfies the three conditions (a), (b) and (c) in Proposition 3.35 when L_3 is substituted for L_m . Therefore, M is L_3 -saturated by the proposition.

$\Rightarrow$ (Only-if). Suppose M is L_3 -saturated. By Proposition 3.35, M is a 3-valued model of Π . It follows from Definition 4.1 that M is also a model of $\frac{\Pi}{M}$. Let $M' = \langle T', F' \rangle$ be the least Herbrand model of $\frac{\Pi}{M}$. (It exists by Proposition 4.7.) Then $M' \preceq M$, i.e. $T' \subseteq T$ and $F' \supseteq F$. For any $b \in T$, $\frac{\Pi}{M} \models_{L_3} b$ by Lemma 3.28 and 4.9 because $M \uplus \Pi \vdash_{L_3} b$. Thus $b \in T'$ and $T = T'$. For any $b \in F'$ and any $H_b = \langle P_b, N_b \rangle \Delta_{\Pi}$ such that $H_b \uplus \Pi \vdash_{L_3} b$, there exist $n_1, \dots, n_q \in N_b$ such that $\Pi \vdash_{L_3} b \leftarrow \text{not} n_1, \dots, \text{not} n_q$. Since M is also a model of the clause by Lemma 3.28, there exists i such that $n_i \in T$. Then $M \uplus \Pi \vdash_{L_3} n_i$ because M is positively L_3 -founded. Therefore, b is L_3 -false-assumable wrt M , and thus $b \in F$ because M is L_3 -saturated. Hence $F = F'$ and M is a 3-valued stable model of Π .

Proof of (b):

$\Leftarrow$ (If). Suppose M is a 3-valued stable-by-case model of Π . Then M is the least Herbrand model of $\frac{\Pi^*}{M}$. Since the least Herbrand model is unique, M can be constructed the way in the proof of Proposition 4.7 as the least Herbrand model of $\frac{\Pi^*}{M}$. Since $\Pi \vdash_{L_2} \Pi^*$, M satisfies the three conditions (a), (b) and (c) in Proposition 3.35 when L_2 is substituted for L_m . Therefore, M is L_2 -saturated by the proposition.

$\Rightarrow$ (Only-if). Essentially the same as the proof of Only-if of (a) except that L_3 is to be replaced with L_2 .

□

Przymusiński extended the concept of 3-valued stable models from normal programs to disjunctive normal programs (a disjunctive normal program is a disjunctive program such that all the literals in the heads and bodies of clauses are atoms.)

Definition 4.4 Let Π be a disjunctive normal program in a 3-valued language L and I be a Herbrand interpretation of L . I is called a 3-valued stable model if I is a minimal Herbrand model of the extended GL-transformation $\frac{\Pi}{I}$ of Π modulo I .

Recall Definition 4.2. A 3-valued stable model of a normal program is the *least* Herbrand model of the extended GL-transformation while a 3-valued stable model of a disjunctive normal program is a *minimal* Herbrand model of the extended GL-transformation. The next theorem shows that (a) of Theorem 4.10 still holds for disjunctive programs.

Theorem 4.11 *Let Π be a disjunctive normal program and $M = \langle T, F \rangle$ a consistent partial interpretation. M is L_3 -saturated iff M is a 3-valued stable model of Π .*

Proof: Let π be any semantical instance of Π wrt M . Then $\pi \models_{L_3} \Pi$.

$\Leftarrow$ (If). Suppose M is a 3-valued stable model of Π . Then M is a model of $\frac{\pi}{M}$ because the truth value of the head of any clause in $\frac{\pi}{M}$ is the same as the truth value of the head of the corresponding clause in $\frac{\Pi}{M}$. Let M' be the least Herbrand model of $\frac{\pi}{M}$. (M' exists by Proposition 4.7.) Then $M' \preceq M$. Because $\frac{\pi}{M} \models_{L_3} \frac{\Pi}{M}$, M' is also a model of $\frac{\Pi}{M}$. Because M is a minimal model of $\frac{\Pi}{M}$ and $M' \preceq M$, $M' = M$. Therefore, M is a 3-valued stable model of $\frac{\pi}{M}$. By Theorem 4.10, M is L_3 -saturated of π . Thus, M is L_3 -saturated of Π .

$\Rightarrow$ (Only-if). Suppose M is L_3 -saturated of Π . The M is also L_3 -saturated of π . M is a 3-valued stable model of π by Theorem 4.10. M is the least Herbrand model of $\frac{\pi}{M}$ and thus a model of $\frac{\Pi}{M}$. Let M' be a model of $\frac{\Pi}{M}$ such that $M' \preceq M$. Let $\frac{\pi'}{M'}$ be a semantical instance of $\frac{\Pi}{M}$ wrt M' such that π' is also a semantical instance of Π wrt M . (Such π' always exists.) Then M' is also a model of $\frac{\pi'}{M'}$ since the truth value of the head of every clause in $\frac{\pi'}{M'}$ is the same as the truth value of the head of the corresponding clause in $\frac{\Pi}{M}$. Since M is the least Herbrand model of $\frac{\pi'}{M'}$, $M \preceq M'$. Therefore, M is a minimal Herbrand model of $\frac{\Pi}{M}$, and thus a 3-valued stable model of Π . $\square$

4.4 The Well-founded Model Semantics

This section shows the equivalence of the well-founded semantics, proposed by Van Gelder, Ross and Schlipf [23], and $\mathcal{L}$ -well-founded semantics.

Definition 4.5 The well-founded model $M_{wf\Pi}$ of a normal program Π is defined as the $\subseteq$ -smallest 3-valued stable model (smallest wrt the F-ordering $\subseteq$) of Π .

The well-founded model was first defined constructively by Van Gelder, Ross and Schlipf [23]. Refer to [23] for their original definitions. The equivalence of Definition 4.5 with the original definitions have been shown in [72] and [73]. Definition 4.5

is presented above to simplify our presentation. (It will follow from Theorem 4.10 and 4.12 that each normal program has at least one 3-valued stable model and furthermore, that each normal program has the $\subseteq$ -smallest 3-valued stable model (wrt the F-ordering $\subseteq$)).

The well-founded semantics is not capable of proving by cases. Since $\vdash_{L_2}$ is capable of proving by cases, we introduce the following definition, based on $\vdash_{L_2}$.

Definition 4.6 Let Π be a program and $\Pi^* = \{\alpha : \Pi \vdash_{L_2} \alpha\}$. The *well-founded-by-case model* M_{wfbc} of Π is defined as the $\subseteq$ -smallest 3-valued stable-by-case model of Π , i.e. the well-founded model of Π^* .

The well-founded-by-case model of Π is first proposed by Schlipf in [80]. Refer to [80] for their original definitions.

Since the well-founded model is defined as the $\subseteq$ -least 3-valued stable model and the well-founded-by case model is defined as the $\subseteq$ -least 3-valued stable-by-case model, we have the following theorem.

Theorem 4.12

- (a) M_{L_3-wf} is the well-founded model;
- (b) M_{L_2-wf} is the well-founded-by-case model.

Proof: The result follows from Theorem 3.26 and Theorem 4.10. $\square$

4.5 The Stable Model Semantics

This section shows the equivalence of the stable model semantics, proposed by Gelfond and Lifschitz [24], and $\mathcal{L}$ -stable semantics.

Definition 4.7 Let Π be a program in a language $\mathbf{L}$ and M be a subset of the Herbrand base $B_{\mathbf{L}}$. The *GL-transformation of Π modulo M* , denoted as $\frac{\Pi}{M}$, is the set obtained from the Herbrand instantiation $[\Pi]_{\mathbf{L}}$ of Π by deleting

- (a) each instantiated clause that has a formula **nota** in its body with $a \in M$, and
- (b) all formulas of the form **notb** in the bodies of the remaining instantiated clauses.

Clearly, $\frac{\Pi}{M}$ is **not** free by Definition 4.7. It follows from the proof of Proposition 4.17 that $\frac{\Pi}{M}$ has the least Herbrand model. Let $M_{\frac{\Pi}{M}}$ be the set representation of the least Herbrand model of $\frac{\Pi}{M}$.

Definition 4.8 Let Π be a program in a language $\mathbf{L}$ and M be a subset of the Herbrand base $B_{\mathbf{L}}$. M is called a *stable set* of Π if M coincides with the least Herbrand model of the GL-transformation $\frac{\Pi}{M}$ of Π modulo M .

We can easily see the analogy between a 3-valued stable model defined in Definition 4.2 and a stable set. Since a stable set corresponds to the least Herbrand model of $\frac{\Pi}{M}$, it is also called a stable model.

Proposition 4.13 Let Π be a program in a 3-valued language $\mathbf{L}$ and I be a 2-valued Herbrand interpretation of $\mathbf{L}$. I is a stable model of Π iff I is a 3-valued stable model of Π .

Proof: Let $M = \{a : /a/_I = \mathbf{t}\}$. I is a Herbrand model of the GL-transformation of Π modulo M iff I is a Herbrand model of the extended GL-transformation of Π modulo I . The result follows immediately from Definition 4.8 and 4.2. $\square$

We now define stable models using $\vdash_{L_2}$. The stable-by-case model of Π was first proposed by Schlipf in [80]. Refer to [80] for its original definition.

Definition 4.9 Let Π be a program and $\Pi^* = \{\alpha : \Pi \vdash_{L_2} \alpha\}$. A stable-by-case model of Π is defined as a 3-valued stable-by-case model of Π , which is total, i.e. a stable model of Π^* .

We have the following theorem.

Theorem 4.14 Let Π be a program and $M = \langle T, F \rangle$.

- (a) M is L_3 -stable model of Π iff M is a stable model of Π ;
- (b) M is L_2 -stable model of Π iff M is a stable-by-case model of Π .

Proof: It follows from Proposition 4.13 that a stable model and a stable-by-case model are a total 3-valued stable model and a total 3-valued stable-by-case model, respectively. The result follows from Theorem 4.10. $\square$

4.6 Semantics Based on AE logic

The framework of Chapter 3 is presented independently of the underlying logic $\mathcal{L}$ although specific properties of the framework are proved when the underlying logic is Lukasiewiczian many-valued logic. In this section, we will show how autoepistemic logic is used in our framework. We will also show how semantics for logic programming based on autoepistemic logic fit in our framework.

4.6.1 Autoepistemic Logic

Autoepistemic (AE) logic is intended to model the beliefs of an ideal rational agent reflecting upon his own belief [61]. The beliefs of the agent are assumed to be expressed as a set of sentences in some logical language augmented by a modal operator L . The intended meaning of $L\phi$ is that ϕ is one of the agent's beliefs. The primary focus of AE logic is that given an initial set of beliefs Π of an agent, what set E of AE logic sentences should be interpreted as the total beliefs of the agent if the agent is introspectively ideal and rational? The truth of an agent's beliefs, expressed as a set of AE logic sentences, will be determined by (1) which sentences not containing L are true in the actual world the agent is in and (2) which sentences the agent believes. A sentence $L\phi$ is true wrt an agent if and only if ϕ is one of his beliefs, i.e. in his set of beliefs. If Π is the agent's beliefs about the actual world, i.e. Π does not contain the self-belief operator, then clearly an ideal agent should believe all of the logical consequences of his initial beliefs, a condition sometimes referred to as *logical omniscience*. On the other hand, a rational agent should not believe any sentences about the world which are not implied by his initial beliefs. Such an argument is still applicable even if Π contains the self-belief operator L . However, the presence of L complicates matters. Because the intended meaning of $L\phi$ depends on the agent's set of beliefs, the definition of set of beliefs becomes circular, which necessitates the use of a fixed-point equation to define the total belief set E . In the following, we will quote Moore's original definition [61]. Alternative formulations for the definition of belief set can be found in [29].

We begin with the definition of the language of AE logic. The base language of the original AE logic defined by Moore is a propositional language. We will adopt Konolige's extension of the base language to a first order one [29]. We assume that the base language is the classical first order language, i.e. the language defined in Definition 2.4 with truth value t and f . The language $\mathbf{L}$ of an autoepistemic logic consists of the set of all formulas formed by all the normal formation rules for a first order language defined in Definition 2.3 and the following rule.

- (e) If F is a sentence, i.e. a formula containing no free variable, then so is LF .

A formula $L\phi$ is called a *modal atom*. A formula formed without using rule (e) is called *objective*. Clearly an objective formula does not contain the modal operator L . Note that nestings such as $LL\phi$ are modal atoms. Because the modal operator only operates on sentences which contain no free variables, there is no quantifying into the scope of a modal atom, e.g., $\exists x Lp(x)$ is not allowed.

The semantics of AE logic is given by stable expansions. Let Π be a set of sentences

of an AE logic. A set E of sentences is a *stable expansion* of Π if

$$E = th(\Pi \cup \{L\phi : \phi \in E\} \cup \{\neg L\phi : \phi \notin E\}).$$

where $th(X)$ denotes the set of all logical consequences of X in the classical first order logic. I.e., $th(X) = \{\phi : X \models_{\mathcal{CL}} \phi\}$, where each modal atom appearing in X or ϕ is considered as a new proposition symbol, or a 0-ary predicate of the language. Detailed discussion about the semantics of AE logic and about stable expansion is beyond the scope of this thesis, and can be found in [61] and [29].

Clearly, if all sentences in Π are objective, then

1. Π has exactly one stable expansion E ; and
2. an objective sentence is in E iff it is a logical consequence of Π .

4.6.2 $\mathcal{AE}$ -Preferable Semantics

With the autoepistemic logic being the underlying logic of logic programs, a negative literal **nota** in a program Π is considered as a negative modal literal $\neg La$. For a set S of ground program clauses, we write $S(L)$ to denote the set obtained from S by replacing every negative literal **nota** with $\neg La$. $[\Pi]_L$ is the propositional representation of a program Π . We write $\Pi(L)$ to denote $[\Pi]_L(L)$, i.e. the set obtained from $[\Pi]_L$ by replacing every negative literal **nota** with $\neg La$. $\Pi(L)$ is a set of sentences of AE logic. The semantics for the logic program Π is given by the semantics of $\Pi(L)$ as a set of AE logic sentences. In the following, we will use $\mathcal{AE}$ to denote the autoepistemic logic and show that such a semantics for logic programming is actually equivalent to the $\mathcal{AE}$ -preferable semantics.

Now we proceed to define the hypothesis incorporating operator $\uplus$ when the underlying logic $\mathcal{L}$ is $\mathcal{AE}$, and the logical implication $\models_{\mathcal{AE}}$. Then for any $H \subseteq \Delta_L$ and a program Π , $H \uplus \Pi$ is defined as the expression $(H \uplus \Pi)(L)$. (I.e., first incorporating H , then replacing **nota** with $\neg La$.) Because the semantics of a set S of sentences of $\mathcal{AE}$ is given by the stable expansions of S , we write $S \models_{\mathcal{AE}} \phi$ to mean that the sentence ϕ is in every stable expansion of S .

With $\uplus$ and $\models_{\mathcal{AE}}$ having been defined, all the concepts and notions defined in Chapter 3 for the logic $\mathcal{L}$ are applicable.

Lemma 4.15, *Let Π be a set of ground normal program clauses, E a set of sentences of $\mathcal{AE}$. The following conditions are equivalent.*

- (a) E is a stable expansion of $\Pi(L)$;

- (b) E is a stable expansion of $\langle \emptyset, N \rangle \uplus \Pi$, where $N \subseteq \{a : a \notin E \cap B_L\}$; and furthermore E is the only stable expansion of $\langle \emptyset, N \rangle \uplus \Pi$ when $N = \{a : a \notin E \cap B_L\}$;
- (c) E is a stable expansion of $\frac{\Pi}{T}$, where $T = \{a : a \in E \cap B_L\}$.
(where $\frac{\Pi}{T}$ is the GL-transformation of Π modulo T .)

Proof: Let $H = \langle \emptyset, N \rangle$. It suffices to show the following are equivalent when $N = \{a : a \notin E \cap B_L\}$.

$$\Pi(L) \cup \{L\phi : \phi \in E\} \cup \{\neg L\phi : \phi \notin E\}$$

and

$$H \uplus \Pi \cup \{L\phi : \phi \in E\} \cup \{\neg L\phi : \phi \notin E\}$$

and

$$\frac{\Pi}{T} \cup \{L\phi : \phi \in E\} \cup \{\neg L\phi : \phi \notin E\}$$

The set $\{L\phi : \phi \in E\} \cup \{\neg L\phi : \phi \notin E\}$ contains La for each $a \in T$ and $\neg La$ for each $a \notin T$ or $a \in N$. In the presence of these modal literals, $\Pi(L)$, $H \uplus \Pi$ and $\frac{\Pi}{T}$ are equivalent. (The equivalence of (a) and (c) has been proved in [24].) $\frac{\Pi}{T}$ is a subset of $H \uplus \Pi$. Thus every $a \in T$ is a logical consequence of $H \uplus \Pi$ since it is a logical consequence of $\frac{\Pi}{T}$. Let E' be a stable expansion of $H \uplus \Pi$. Then $T \subseteq E'$ because every $a \in T$ is a logical consequence of $H \uplus \Pi$. Thus E' contains La for each $a \in T$. In the presence of these literals, $H \uplus \Pi$ and $\frac{\Pi}{T}$ are equivalent. Hence, E' is also a stable expansion of $\frac{\Pi}{T}$. Because $\frac{\Pi}{T}$ is objective and has exactly one stable expansion, $E' = E$, and E is the only stable expansion of $H \uplus \Pi$. $\square$

With $\mathcal{AE}$ being the underlying logic, a logic program Π is translated into an autoepistemic theory $\Pi(L)$. Since the semantics of $\mathcal{AE}$ is specified in terms of stable expansions, we need only consider autoepistemic theories which do have stable expansions.

Theorem 4.16 *Let Π be a program such that $\Pi(L)$ has at least one stable expansion, and $M = \langle T, F \rangle$ be a partial interpretation. The following conditions are equivalent.*

- (a) M is an $\mathcal{AE}$ -preferable model of Π ;
- (b) There exists a stable expansion E of $\Pi(L)$ such that $T = E \cap B_L$ and $F = \{b : b \notin E \cap B_L\}$;
- (c) M is a stable model of Π .

Proof: (a) $\Rightarrow$ (b): Let $M = \langle T, F \rangle$ be $\mathcal{AE}$ -preferable of Π . $M \uplus \Pi$ has a stable expansion. (Otherwise, for every literal $l \in Lit_{\mathbf{L}}$, $M \uplus \Pi \models_{\mathcal{AE}} l$. If $M \neq \langle \emptyset, \emptyset \rangle$, H is not $\mathcal{AE}$ -noncontradictory. If $H = \langle \emptyset, \emptyset \rangle$, $H \uplus \Pi$ is the same as Π . Both cases contradict the premises of the theorem.) Let E be a stable expansion of $M \uplus \Pi$, $T' = E \cap B_{\mathbf{L}}$, $F' = \{b : b \notin E \cap B_{\mathbf{L}}\}$ and $M' = \langle T', F' \rangle$. $M \uplus \Pi \models_{\mathcal{AE}} a$ for each $a \in T$ because H is positively $\mathcal{AE}$ -founded. Thus, $T \subseteq T'$. We then show $F \subseteq F'$. If not, then there exists an atom a in $F - F'$ such that $a \in E$. By Lemma 4.15, E is the only stable expansion of $\langle \emptyset, F' \rangle \uplus (M \uplus \Pi)$, which can be expressed as $\langle T, F' \cup F \rangle \uplus \Pi$. Thus $\langle T, F' \cup F \rangle \uplus \Pi \models_{\mathcal{AE}} a$. Because $a \in F$ is $\mathcal{AE}$ -false-assumable wrt M , there exists $b \in F' \cup F$ such that $H \uplus \Pi \models_{\mathcal{AE}} b$. Because M is $\mathcal{AE}$ -noncontradictory, $b \notin F$. Since E is a stable expansion of $M \uplus \Pi$ and b is in every stable expansion of $M \uplus \Pi$, $b \notin F'$. Therefore, $b \notin F' \cup F$. Contradiction. Hence, $F \subseteq F'$. Then E is the only stable expansion of $M' \uplus \Pi$. By Lemma 4.15 again, E is a stable expansion of $\Pi(\mathbf{L})$.

We now prove $M = \langle T', F' \rangle$ by showing that $\langle T', F' \rangle$ is $\mathcal{AE}$ -admissible of Π . Because E is the only stable expansion of $M' \uplus \Pi$, $M' \uplus \Pi \models_{\mathcal{AE}} a$ for each $a \in T'$ and $M' \uplus \Pi \not\models_{\mathcal{AE}} a$ for any $a \in F'$. Thus M' is positively $\mathcal{AE}$ -founded and $\mathcal{AE}$ -noncontradictory of Π . For any $a \in F'$ and any $H_a = \langle P_a, N_a \rangle \subseteq \Delta_{\mathbf{L}}$ such that $H_a \uplus \Pi \models_{\mathcal{AE}} a$, $H_{aN} \uplus \Pi \models_{\mathcal{AE}} a$, where $H_{aN} = \langle \emptyset, N_a \rangle$. If there exists $b \in N_a$ such that $b \in T'$, then $M' \uplus \Pi \models_{\mathcal{AE}} b$. Then a is $\mathcal{AE}$ -false-assumable wrt M' . If there does not exist such an atom b , then $N_a \subseteq F'$. By Lemma 4.15, E is a stable expansion of $H_{aN} \uplus \Pi$. Because $H_{aN} \uplus \Pi \models_{\mathcal{AE}} a$, $a \in E$. This contradicts the fact that a is in F' . Therefore, M' is negatively $\mathcal{AE}$ -founded of Π . Hence, M' is $\mathcal{AE}$ -admissible of Π . Since M is $\mathcal{AE}$ -preferable and $M \subseteq \langle T', F' \rangle$, $M = \langle T', F' \rangle$.

(b) $\Rightarrow$ (c): Let E be a stable expansion of $\Pi(\mathbf{L})$, $T = E \cap B_{\mathbf{L}}$ and $F = \{b : b \notin E \cap B_{\mathbf{L}}\}$. By Lemma 4.15, E is the only stable expansion of $\frac{\Pi}{T}$ since $\frac{\Pi}{T}$ is objective. Let $N \subseteq B_{\mathbf{L}}$ be the least Herbrand model of $\frac{\Pi}{T}$. For any $a \in B_{\mathbf{L}}$, $a \in T$ iff $a \in E$ iff $\frac{\Pi}{T} \models_{\mathcal{CL}} a$ iff $a \in N$ by Theorem 4.18. Thus $T = N$. Hence T is a stable set of Π and M is a stable model of Π .

(c) $\Rightarrow$ (a): Let $M = \langle T, F \rangle$ be a stable model of Π . Suppose E is the only stable expansion of $\frac{\Pi}{T}$. Then by Theorem 4.18, for any $a \in B_{\mathbf{L}}$, $a \in T$ iff $a \in E$, and $a \in F$ iff $a \notin E$. By Lemma 4.15, E is the only stable expansion of $\langle T, F \rangle \uplus \Pi$. By the same proof in (a) $\Rightarrow$ (b), we can show that M is an $\mathcal{AE}$ -admissible model of Π . Because M is total, it is a maximal $\mathcal{AE}$ -admissible model. Hence M is $\mathcal{AE}$ -preferable of Π . $\square$

By this theorem, the semantics based on AE logic is actually the $\mathcal{L}$ -preferable semantics when the underlying logic $\mathcal{L}$ is the autoepistemic logic $\mathcal{AE}$. Furthermore, if a program has a unique stable model, then it has a unique $\mathcal{AE}$ -preferable model. The stable model semantics can also be characterized by our framework by choosing

the underlying logic as $\mathcal{AE}$.

4.7 New Semantics

In the above sections, we presented various semantics for (disjunctive) normal programs and showed that each one of the semantics is equivalent to a semantics defined in the framework proposed in Chapter 3. We claim that the following semantics are new semantics first proposed in this dissertation.

L_2 -preferable semantics is new for both normal and disjunctive normal programs; and

L_2 -well-founded, L_2 -saturated and L_2 -stable semantics are new for disjunctive normal program.

(The original definitions of the well-founded-by-case and stable-by-case model by Schlipf in [80] are not applicable to disjunctive programs. Refer to [80] for their original definitions.) The common advantage of these new semantics over the existing ones is that the new ones are all capable of proving by cases due to the choice of the underlying logic L_2 . In the rest of this section, we analyze two examples to compare various semantics.

Although the underlying logic for the preferential extension semantics is the classical logic, the reasoning power of the logic is reduced and the original semantics of the logic is altered due to the introduction of abducible atoms. The following Example 4.2 will indicate that the reasoning-by-case capability of the classical logic is lost when abducible atoms are introduced. The semantics is equivalent with the L_3 -preferable semantic, as indicated by Theorem 4.3.

Example 4.2 Consider the following two programs.

$\Pi_7 :$	$a \leftarrow \text{not} b$	$\Pi_8 :$	$a \leftarrow \text{not} b$
	$b \leftarrow \text{not} a$		$b \leftarrow \text{not} c$
	$d \leftarrow a$		$c \leftarrow \text{not} a$
	$d \leftarrow b$		$d \leftarrow a$
	$e \leftarrow \text{not} d$		$d \leftarrow b$
			$d \leftarrow c$
			$e \leftarrow \text{not} d$

Π_7 has two preferential extensions E_1 and E_2 , where E_1 contains $\{d, a, \bar{e}, \bar{b}\}$ and E_2 contains $\{d, b, \bar{e}, \bar{a}\}$. Π_7 has two L_2 -preferable models. One is $M_1 = \langle \{d, a\}, \{e, b\} \rangle$.

The other is $M_2 = \langle \{d, b\}, \{e, a\} \rangle$. So for Π_7 , the L_2 -preferable semantics and the preferential extension semantics coincides with each other, and d and $\neg e$ can be inferred from Π_7 according to either semantics. However, we do not have the coincidence for a very similar program Π_8 . Since there does not exist a nonempty set of abducible atoms such that every abducible atom in the set is acceptable wrt the set for Π_8 , neither d nor $\neg e$ may be inferred from Π_8 according to the preferential extension semantics. The only preferential extension of Π_8 is given by $\bar{\Pi}_8 \cup \emptyset$. Π_8 has one L_2 -preferable model $M_3 = \langle \{d\}, \{e\} \rangle$. So both d and $\neg e$ can be inferred from Π_8 according to the L_2 -preferable semantics. We believe that d and e should have the same semantics in both Π_7 and Π_8 when the underlying logic is the classical logic. d is to be inferred from both program because d can be derived from both program by reasoning by cases. This example also indicates that the preferential extension semantics does not contain all conclusions reachable from reasoning by cases although it sometimes does for a program such as Π_7 . This result is not surprising because by Theorem 4.3, the semantics coincides with L_3 -preferable semantics and the logic L_3 is not capable of reasoning by cases in general.

Let us consider the difference of acceptability of an abducible atom and L_2 -false-assumability of an (ordinary) atom. Let $H = \emptyset$. e is L_2 -false-assumable of Π_7 wrt H since $\Pi \models_{L_2} d$ and any hypothesis set that predicts e must include $\neg d$. However, the corresponding abducible atom of e is not acceptable wrt $\bar{H}$.

Example 4.3 Consider the following two programs.

$\Pi'_7 :$	$a \leftarrow \text{not} b$	$\Pi'_8 :$	$a \leftarrow \text{not} b$
	$b \leftarrow \text{not} a$		$b \leftarrow \text{not} c$
	$u, v \leftarrow a$		$c \leftarrow \text{not} a$
	$u, v \leftarrow b$		$u, v \leftarrow a$
			$u, v \leftarrow b$
			$u, v \leftarrow c$

Π'_7 has the following 3-valued stable models, or L_3 -saturated models. $M_0 = \langle \emptyset, \emptyset \rangle$, $M_1 = \langle \{u, a\}, \{v, b\} \rangle$, $M_2 = \langle \{v, a\}, \{u, b\} \rangle$, $M_3 = \langle \{u, b\}, \{v, a\} \rangle$ and $M_4 = \langle \{v, b\}, \{u, a\} \rangle$. M_i , $1 \leq i \leq 4$, are L_3 -preferable of Π'_7 . Consider L_2 -saturated models of Π'_7 . M_i , $1 \leq i \leq 4$, are L_2 -saturated models of Π'_7 . It is worthy of note that M_0 is not an L_2 -saturated model of Π'_7 . Under both L_3 -preferable and L_2 -saturated semantics, Π'_7 implies $u \vee v$. Now let us consider Π'_8 , which is similar to Π'_7 . The only 3-valued stable, or L_3 -saturated model of Π'_8 is M_0 . Under the L_3 -preferable semantics, Π'_8 does not implies $u \vee v$. However, Π'_8 implies $u \vee v$ under the L_2 -saturated semantics because the only L_2 -saturated models of Π'_8 are $N_1 = \langle \{u\}, \{v\} \rangle$ and $N_2 = \langle \{v\}, \{u\} \rangle$.

A shortcoming of the well-founded model semantics is that the semantics can not handle floating conclusions, i.e. conclusions which can only be reached by proving by cases. Another criticism against the semantics is that "too many" literals are undefined under the semantics. It has been pointed out that the undefinedness of a semantics should be minimized [93]. Although it overcomes the above weakness of the well-founded semantics, the stable model semantics is not applicable to every program. The new semantics, L_2 -preferable semantics, is capable of handling floating conclusions, has minimized undefinedness, and is applicable to every disjunctive normal program. The framework presents an overview of most of the existing semantics. Since the new semantics is missing from the whole picture, the framework also provides justification for the semantics.

4.8 The Least Herbrand Model Semantics for Definite Programs

This section presents the least Herbrand model semantics for the class of definite programs for the purpose of reference. Readers familiar with the results are encouraged to skip this section. The least Herbrand model semantics for the class of definite programs has played a central role in the theory of logic programming. Although various results about the least Herbrand model of a definite program have been presented in the context of classical logic [42]. We show that the least Herbrand model is independent of the truth value set of the language. We redefine necessary concepts and present some existing results in the context of multivalued logic L_m .

Definition 4.10 Let L be a first order language and I_1 and I_2 be two Herbrand interpretations of L . Then the *intersection* I of I_1 and I_2 , denoted by $I_1 \dot{\cap} I_2$, is the Herbrand interpretation obtained from I_1 and I_2 by specifying that for each n -ary predicate p of L , the mapping p' assigned to p is the following:

$$p'(t_1, \dots, t_n) = \min\{p'_1(t_1, \dots, t_n), p'_2(t_1, \dots, t_n)\},$$

where each $t_i \in B_L$ and p'_1 and p'_2 are the mappings assigned to p by I_1 and I_2 respectively.

Example 4.4 Let L be a 2-valued first order language. Then every Herbrand interpretation of L is 2-valued. A Herbrand interpretation I of L can be identified by a subset, say T , of the Herbrand base by letting $T = \{a \in B_L : /a/_I = t\}$. Conversely, given an arbitrary subset T of the Herbrand base, then a corresponding Herbrand interpretation can be specified according to T . I.e. the interpretation maps each

ground atom a to $\mathbf{t}$ if $a \in T$, otherwise to $\mathbf{f}$. With this identification of a Herbrand interpretation of $\mathbf{L}$ as a subset of the Herbrand base, the concept of the intersection of two Herbrand interpretation coincides with the concept of set intersection. Specifically, let I_1 and I_2 be two Herbrand interpretations of $\mathbf{L}$ and T_1 and T_2 be the two subsets of $B_{\mathbf{L}}$ identifying I_1 and I_2 respectively. A subset T of $B_{\mathbf{L}}$ identifies $I_1 \hat{\cap} I_2$ iff $T = T_1 \cap T_2$.

Example 4.5 Let $\mathbf{L}$ be a 3-valued first order language. As stated before, there is a one - one correspondence between Herbrand interpretations of $\mathbf{L}$ and consistent partial interpretations. Let I_1 and I_2 be two Herbrand interpretations of $\mathbf{L}$ and $\langle T_1, F_1 \rangle$ and $\langle T_2, F_2 \rangle$ be the two corresponding partial interpretations. Then a consistent partial interpretation $\langle T, F \rangle$ corresponds to $I_1 \hat{\cap} I_2$ iff $\langle T, F \rangle = \langle T_1 \cap T_2, F_1 \cup F_2 \rangle$.

Proposition 4.17 (*Model intersection property*)

Let Π be a definite program and $\{M_i\}_{i \in I}$ be a nonempty set of Herbrand models for Π . Then $\hat{\cap}_{i \in I} M_i$ is a Herbrand model for Π .

Proof: Let M be the Herbrand interpretation $\hat{\cap}_{i \in I} M_i$. It suffices to show that M is a model for the Herbrand instantiation $[\Pi]_{\mathbf{L}}$. For any instantiated clause $\alpha : a \leftarrow a_1, \dots, a_n$ in $[\Pi]_{\mathbf{L}}$. Then $/a/M = \min\{/a/M_i : i \in I\}$.

$$\begin{aligned} /(a_1 \wedge \dots \wedge a_n)/M &= \min\{/a_j/M : j \leq n\} \\ &= \min\{\min\{/a_j/M_i : i \in I\} : j \leq n\} \\ &= \min\{/a_j/M_i : i \in I, j \leq n\} \end{aligned}$$

Because for each $i \in I$, $/a/M_i \geq \min\{/a_j/M_i : j \leq n\}$, $/a/M \geq \min\{/a_j/M_i : i \in I, j \leq n\}$. Therefore, $/\alpha/M = \mathbf{t}$. $\square$

Since the Herbrand interpretation that maps every ground atom to the truth value $\mathbf{t}$ is a Herbrand model for any definite program, the set of all Herbrand models for a definite program Π is not empty. Thus the intersection of all Herbrand models for Π , which is also a model for Π by Proposition 4.17, is actually the *least Herbrand model* for Π .

The next theorem shows that in the least Herbrand model, the ground atoms denoting $\mathbf{t}$ are precisely those which are logical consequences of the program while the ground atoms denoting $\mathbf{f}$ are precisely those which are not logical consequences of the program, or those the negation of which can be inferred under the closed world assumption. It also shows that the least Herbrand model is 2-valued. This result is first given by van Emden and Kowalski [14], but only for 2-valued languages.

Theorem 4.18 *Let M be the least Herbrand model of a definite program Π in a language $\mathbf{L}$. Then for any ground atom $a \in B_{\mathbf{L}}$,*

$$/a/M = \mathbf{t} \text{ iff } \Pi \models_{\mathcal{L}} a$$

$$/a/M = \mathbf{f} \text{ iff } \Pi \not\models_{\mathcal{L}} a$$

Proof: It suffices to show that for any ground atom $a \in B_{\mathbf{L}}$, $\Pi \models_{\mathcal{L}} a$ implies $/a/M = \mathbf{t}$ and $\Pi \not\models_{\mathcal{L}} a$ implies $/a/M = \mathbf{f}$. If $\Pi \models_{\mathcal{L}} a$, then for every Herbrand model N for Π , $/a/N = \mathbf{t}$. Thus $/a/M = \mathbf{t}$. If $\Pi \not\models_{\mathcal{L}} a$, there exists a model N for Π such that $/a/N \neq \mathbf{t}$. By Proposition 2.2, there exists a Herbrand model N' such that $/a/N' \neq \mathbf{t}$. We then define a Herbrand interpretation I as follows. For each n -ary predicate p of $\mathbf{L}$, the mapping p' assigned to p is the following: for $(t_1, \dots, t_n) \in U_{\mathbf{L}}^n$,

$$p'(t_1, \dots, t_n) = \begin{cases} \mathbf{t} & \text{whenever } /(p(t_1, \dots, t_n))/N' = \mathbf{t}; \\ \mathbf{f} & \text{otherwise.} \end{cases}$$

Thus for any ground atom $b \in B_{\mathbf{L}}$, $/b/I = \mathbf{t}$ whenever $/b/N' = \mathbf{t}$; otherwise, $/b/I = \mathbf{f}$.

Next we show that I is a model for the Herbrand instantiation $[\Pi]_{\mathbf{L}}$ of Π . For any instantiated clause $\alpha : a \leftarrow a_1, \dots, a_n$, if $/(a_1 \wedge \dots \wedge a_n)/I = \mathbf{t}$, then $/(a_1 \wedge \dots \wedge a_n)/N' = \mathbf{t}$. Since N' is a model for α , $/a/N' = \mathbf{t}$ by Proposition 2.5. Thus by the definition of I , $/a/I = \mathbf{t}$. Therefore, $/\alpha/I = \mathbf{t}$. By Proposition 2.4, I is a Herbrand model for Π . And by the definition of I , $/a/I = \mathbf{f}$ since $/a/N' \neq \mathbf{t}$. Thus, $/a/M = \mathbf{f}$. $\square$

Since by Theorem 4.18, the least Herbrand model of a definite program is 2-valued, there is a total interpretation corresponding to it. We use $\langle M_{\Pi}, N_{\Pi} \rangle$ to denote this total interpretation. Then $\langle M_{\Pi}, N_{\Pi} \rangle$ represents the least Herbrand model semantics for definite programs. Or formally,

Definition 4.11 *The least Herbrand model semantics SEM_{IH} for the class of definite programs is defined as follows:*

$$SEM_{IH}(\Pi) = \langle M_{\Pi}, N_{\Pi} \rangle$$

where Π is a definite program.

Let $\mathbf{L}$ be a first order language in which programs are written. Then the power set $2^{B_{\mathbf{L}}}$ of $B_{\mathbf{L}}$ forms a complete lattice with the partial order of set inclusion $\subseteq$. In fact, each element I in $2^{B_{\mathbf{L}}}$ can identify a 2-valued Herbrand interpretation. For an atom $a \in B_{\mathbf{L}}$ we write $I \models a$ when $a \in I$ and $I \models \neg a$ when $a \notin I$. The mapping given in the following definition plays a very important role in the theory of logic programming. The next definition was first given in [14].

Definition 4.12 Let Π be a normal program of a language $\mathbf{L}$. The mapping $T_\Pi : 2^{B_\mathbf{L}} \mapsto 2^{B_\mathbf{L}}$ is defined as follows. Let I be a subset of $B_\mathbf{L}$. Then

$$T_\Pi(I) = \{a \in B_\mathbf{L} : a \leftarrow l_1, \dots, l_n \in [\Pi]_\mathbf{L} \text{ and } I \models l_i, 1 \leq i \leq n\}$$

where $[\Pi]_\mathbf{L}$ is the Herbrand instantiation of Π .

Clearly, T_Π is monotonic when Π is a definite program. It can actually be defined as

$$T_\Pi(I) = \{a \in B_\mathbf{L} : a \leftarrow a_1, \dots, a_n \in [\Pi]_\mathbf{L} \text{ and } \{a_1, \dots, a_n\} \subseteq I\}.$$

We are only interested in definite programs in this section, but we need Definition 4.12 for future use. T_Π provides the link between the declarative and procedural semantics of a definite program Π [42].

Proposition 4.19 [42, Proposition 6.3] *Let Π be a definite program. Then the mapping T_Π is continuous.*

The next theorem provides a fixed point characterization of the least Herbrand model of a definite program. The theorem is a major result due to van Emden and Kowalski [14]. (Only classical logic was in question in [14]. We do not make this restriction here.)

Theorem 4.20 *Let Π be a definite program of a language $\mathbf{L}$. Then $M_\Pi = \text{lfp}(T_\Pi) = T_\Pi \uparrow^\omega$.*

Proof: By Proposition 3.15 and Proposition 4.19, $\text{lfp}(T_\Pi) = T_\Pi \uparrow^\omega$. We then prove $T_\Pi \uparrow^\omega = M_\Pi$. We first show by induction on n that for every n , every ground atom in $T_\Pi \uparrow^n$ is a logical consequence of Π . Thus by Theorem 4.18, $T_\Pi \uparrow^n \subseteq M_\Pi$. We need the following facts:

Fact 1: For any program clause $\alpha \in \Pi$ and any instantiated clause $\beta \in [\alpha]_\mathbf{L}$, $\alpha \models_{\mathcal{L}} \beta$ by Proposition 2.5. Furthermore, $\Pi \models_{\mathcal{L}} \beta$;

Fact 2: If $\Pi \models_{\mathcal{L}} a_0 \leftarrow a_1, \dots, a_m$ and $\Pi \models_{\mathcal{L}} a_1 \wedge \dots \wedge a_m$, then $\Pi \models_{\mathcal{L}} a_0$, where each $a_j \in B_\mathbf{L}$, by the definition of $\leftarrow$.

When $n = 0$, $T_\Pi \uparrow^0 = \emptyset$. Therefore, $T_\Pi \uparrow^n \subseteq M_\Pi$.

Assume when $n = k$, $T_\Pi \uparrow^n \subseteq M_\Pi$.

Consider $n = k + 1$. For any $a \in T_\Pi \uparrow^{k+1}$, there exists an instantiated clause $\beta : a \leftarrow a_1, \dots, a_m$ of Π such that $\{a_1, \dots, a_m\} \subseteq T_\Pi \uparrow^k$. By the inductive assumption, $\Pi \models_{\mathcal{L}} a_1 \wedge \dots \wedge a_m$. By Fact 1, $\Pi \models_{\mathcal{L}} \beta$. By Fact 2, $\Pi \models_{\mathcal{L}} a$. Therefore, $T_\Pi \uparrow^{k+1} \subseteq M_\Pi$. The induction is completed. Hence, $T_\Pi \uparrow^\omega \subseteq M_\Pi$.

We then show that for a ground atom $a \in B_L$, a is not a logical consequence of Π if $a \notin T_\Pi \uparrow^\omega$. We define a Herbrand interpretation I as follows. For each n -ary predicate p of L , the mapping p' assigned to p is the following: for $(t_1, \dots, t_n) \in U_L^n$,

$$p'(t_1, \dots, t_n) = \begin{cases} \mathbf{t}, & \text{whenever } p(t_1, \dots, t_n) \in T_\Pi \uparrow^\omega; \\ \mathbf{f}, & \text{otherwise.} \end{cases}$$

It is straightforward to show that I is a model of the Herbrand instantiation $[\Pi]_L$ of Π because $T_\Pi \uparrow^\omega$ is a fixed point. Then by Proposition 2.4, I is a model of Π and for any ground atom $a \notin T_\Pi \uparrow^\omega$, $a/I = \mathbf{f}$. Thus, $\Pi \not\models_{\mathcal{L}} a$. Hence, $M_\Pi \subseteq T_\Pi \uparrow^\omega$. Thus, $T_\Pi \uparrow^\omega = M_\Pi$. $\square$

4.9 Summary

The saturation of an $\mathcal{L}$ -admissible sets amounts to the maximization (wrt set inclusion) of the sets with its founded hypothesis literals. As indicated by Proposition 3.10, such maximization of the $\mathcal{L}$ -admissible sets preserves the $\mathcal{L}$ -admissibility of the sets. Consequently, we defined the $\mathcal{L}$ -saturated models of programs. Based on the concept of $\mathcal{L}$ -saturation, we introduced the $\mathcal{L}$ -well-founded, $\mathcal{L}$ -saturated, $\mathcal{L}$ -preferable and $\mathcal{L}$ -stable semantics for logic programming. The $\mathcal{L}$ -well-founded model of a program is defined as the least $\mathcal{L}$ -saturated model (wrt the F-ordering $\subseteq$). By Proposition 3.11, the $\mathcal{L}$ -preferable models are the maximal $\mathcal{L}$ -saturated models (wrt the F-ordering $\subseteq$). Furthermore, the $\mathcal{L}$ -stable models are defined as $\mathcal{L}$ -saturated models which are total. Therefore, the hypothetical reasoning approach to the semantics of logic programming introduced in Chapter 3 is based on the saturation of $\mathcal{L}$ -admissible models. Let us call such an approach the $\mathcal{L}$ -saturation approach.

In the previous sections, we reviewed most of the semantics for logic programming. The approach to the semantics is based on the minimization of Herbrand models of the (extended) GL-transformation of a program, as indicated by Proposition 4.8. Let us call such an approach the stablization approach. We have shown that the two distinct approaches — saturation approach and stablization approach are equivalent in that the minimization of Herbrand models of the (extended) GL-transformation of a program or the set of all oriented consequences of a program corresponds to the saturation of $\mathcal{L}$ -admissible models of the program for different choices of the Lukasiewiczian logics. Indeed, each of the existing semantics redefined in Section 4.3 is equivalent with a semantics newly defined in Section 4.1.

This distinction of the saturation approach and the stablization approach has an analog in formal approaches to nonmonotonic reasoning. In the first chapter

of [27], the formal approaches are classified as minimization approaches and proof-based approaches. The minimization approaches include the circumscription [52, 53], which attempts to minimize abnormality, and the closed world assumption [75, 58] for related database applications. In these approaches, the meaning of the theory or databases is given by the *minimal* models of the theory or database. (In the case of circumscription, a model M is "smaller" than another model N if the extents of predicates being minimized, usually the ones expressing abnormality, are "smaller" in M than in N .) The proof-based approaches include default logic [76] and autoepistemic logic [61]. The semantics for a theory of these logics are given by extensions (called expansions in [61]) which are fixed points of some operators. One of the basic ideas for these extensions is that they contain as many sentences as possibly derivable from a given theory and preserve the consistency of the extensions.

The stablization approach to the semantics for logic programming is similar to the minimization approach to nonmonotonic reasoning. The formation or the syntactical form of a program plays an important role in the study of semantics for logic programming, and minimization of models for the program itself would lose the syntactical information embedded in the program. Therefore, direct use of the minimization approach in logic programming is not very useful (except for definite programs). However, with the introduction of the GL-transformation or extended GL-transformation, the approach becomes appealing because the transformation retains the syntactical information embedded in the program. Actually, the stablization of a Herbrand model of a program means the minimization of a Herbrand model of the transformation of the program.

The saturation approach to the semantics for logic programming is based on the framework for hypothetical reasoning, and all the basic concepts or notions for the approach are introduced in terms of the entailment $\models_{\mathcal{L}}$ of a logic $\mathcal{L}$. When $\mathcal{L}$ is a Lukasiewiczian logic, the entailment $\models_{\mathcal{L}}$ is equivalent to the oriented derivation $\vdash_{\mathcal{L}}$ as indicated by Lemma 3.28. Thus, all the concepts or notions can be equivalently defined in terms of $\vdash_{\mathcal{L}}$. An $\mathcal{L}$ -saturated extension contains all the sentences derivable for the corresponding scenario which incorporates as many hypotheses as possible and preserves the $\mathcal{L}$ -admissibility. An $\mathcal{L}$ -saturated model contains both the positive information derivable from the scenario and the negative information incorporated in the scenario as hypotheses. Especially, an $\mathcal{L}$ -preferable extension contains as many sentences as possibly derivable from a scenario and preserves the $\mathcal{L}$ -admissibility of the hypothesis set of the extension. The admissibility implies consistency of the extension and reflects the importance of the formation of logic programs.

The minimization approaches and the proof-based approaches to nonmonotonic reasoning are distinct but also related. In the context of logic programming, the two

distinct approaches — the stablization approach and the saturation approach — are much more closely related as indicated by Theorem 4.10, 4.12 and 4.14. This is because of the special syntactical form, i.e. program clausal form, adopted in logic programming. Actually, various formalizations of nonmonotonic reasoning coincide with each other when the knowledge base is represented as a logic program [68].

We also reviewed two other approaches, autoepistemic logic based and hypothetical reasoning based approaches. In both of the approaches, *nota* is transformed to another form to study the semantics of logic programming. By Theorem 4.3 and 4.16, we show that both approaches are special cases of our $\mathcal{L}$ -saturation based approach.

Chapter 5

Further Analyses: Extended Logic Program

In Chapter 4, we have shown that most existing semantics proposed for (disjunctive) normal programs fit in the framework proposed in Chapter 3. In the chapter, we shall show that semantics proposed for disjunctive extended logic programs fit in our framework by comparing them with various answer set semantics defined in Definition 3.42.

5.1 Answer Set Semantics

5.1.1 GL-Answer Set

Gelfond and Lifschitz extended the stable model semantics proposed for normal programs to disjunctive extended logic programs [25, 26]. We show that the semantics is actually equivalent to L_3 -stable answer set semantics.

A disjunctive extended logic program is a set of formulas of the form $L_1, \dots, L_k \leftarrow L_{k+1}, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$, where L_i 's are literals. Let Π be a disjunctive extended program in a language $\mathbf{L}$ and $S \subseteq \text{Lit}_{\mathbf{L}}$ be a set of literals. The *GL-transformation of Π modulo S* , denoted as $\frac{\Pi}{S}$, is the set obtained from the Herbrand instantiation $[\Pi]_{\mathbf{L}}$ of Π by deleting

- (a) each instantiated clause that has a formula **not** l in its body with $l \in S$, and
- (b) all formulas of the form **not** l in the bodies of the remaining instantiated clauses.

Clearly, $\frac{\Pi}{S}$ is **not** free. In [25, 26], $\frac{\Pi}{S}$ was denoted as Π^S . Since $\frac{\Pi}{S}$ is a straightforward extension of Definition 4.7, we adopt the same notation as in Definition 4.7.

Definition 5.1 Let Π be a disjunctive extended program that does not contain **not**. The *answer set* of Π is the minimal subset S of $Lit_{\mathbf{L}}$ such that

- (a) for any clause $l_1, \dots, l_k \leftarrow l_{k+1}, \dots, l_m$ in $[\Pi]_{\mathbf{L}}$, if $l_{k+1}, \dots, l_m \in S$, then there exists j such that $l_j \in S$, $1 \leq j \leq k$;
- (b) if S contains a pair of complementary literals, the $S = Lit_{\mathbf{L}}$.

For any disjunctive extended program Π (that may contain **not**), S is an *answer set* of Π if S is the answer set of the GL-transformation of Π modulo $S \frac{\Pi}{S}$.

We also refer to an answer set in Definition 5.1 as an GL-answer set. When Π is not disjunctive, a GL-answer set is actually the smallest set satisfying the conditions in Definition 5.1.

Lemma 5.1 Let Π be an extended program that does not contain **not**, and $S \subseteq Lit_{\mathbf{L}}$ be a consistent set of literals. Then S is an answer set of Π if and only if S is an answer set of every semantical instance of Π wrt $H = \langle S, Lit_{\mathbf{L}} - S \rangle$.

Proof: It follows from Definition 3.32 that S satisfies the condition *b* of Definition 5.1 if and only if S satisfies the condition for any semantical instance π wrt H . Then the result follows immediately. $\square$

It is worthy of note that an answer set of an extended program is similar with a stable set of a normal program defined in Definition 4.8. By Theorem 4.20, a stable set is actually an answer set of a normal program, which does not contain $\neg$. Therefore, a stable set may be equivalently redefined in the same way as an answer set. It is no surprise that the above work may be reduced from extended programs to normal programs.

Let Π be an extended program of a language $\mathbf{L}$. For any predicate P occurring in Π , let P' be a new predicate of the same arity. The atom $P'(\dots)$ will be called the positive form of the negative literal $\neg P(\dots)$. Every positive literal is, by definition, its own positive form. The positive form of a literal L will be denoted by L^+ . Π^+ stands for the normal program obtained from Π by replacing every literal with its positive form, i.e., Π^+ is a set of formula of the form

$$L_0^+ \leftarrow L_1^+, \dots, L_m^+, \text{not } L_{m+1}^+, \dots, \text{not } L_{m+n}^+.$$

For any $S \subseteq Lit_{\mathbf{L}}$, S^+ stands for the set of the positive forms of the literals in S . S is said to be *consistent* if it contains no pairs of complementary literals.

Lemma 5.2 [25, 26, Proposition 2] A consistent set $S \subseteq Lit_{\mathbf{L}}$ is an answer set of an extended program Π if and only if S^+ is an answer set of Π^+ .

We now show that the GL-answer set semantics fits in our framework.

Theorem 5.3 *Let Π be a disjunctive extended program, $S \subseteq Lit_{\mathbf{L}}$ be consistent and $H = \langle S, Lit_{\mathbf{L}} - S \rangle$. The following are equivalent.*

- (a) S is a GL-answer set of Π ;
- (b) H is L_3 -stable of Π ;
- (c) S is an L_3 -stable answer set of Π .

Proof: By Lemma 5.1, it suffices to consider Π as an extended program.

(a) $\Leftrightarrow$ (b):

Let $N = Lit_{\mathbf{L}} - S$ and $H^+ = \langle S^+, N^+ \rangle$. Then for any l , $H^+ \uplus \Pi^+ \vdash_{L_3} l^+$ if and only if $H \uplus \Pi \vdash_{L_3} l$. It follows that

Fact: H^+ is L_3 -stable of Π^+ if and only if H is L_3 -stable of Π .

We now show the result.

- S is an answer set of Π ;
- iff S^+ is an answer set of Π^+ by Lemma 5.2;
- iff H^+ is L_3 -stable of Π^+ by Theorem 4.14;
- iff H is L_3 -stable of Π by the above fact.

(b) $\Leftrightarrow$ (c):

The result follows from Definition 3.42 and Proposition 3.32. $\square$

5.1.2 Autoepistemic Representation of GL-Answer Set

Applications of autoepistemic logic theory to the semantics of logic programming have been extended from normal programs to extended programs [6, 40, 49]. Autoepistemic logic is described in section 4.6. A normal logic program is transformed into a theory of AE logic, and the semantics of logic programming is given by the stable expansion semantics of the logic. Recall the equation for the stable expansion E of a set Π of AE logic sentences.

$$E = th(\Pi \cup \{L\phi : \phi \in E\} \cup \{\neg L\phi : \phi \notin E\}),$$

and the transformation as follows. Each clause of a normal program

$$a_0 \leftarrow a_1, \dots, a_m, \text{not} a_{m+1}, \dots, \text{not} a_n$$

is transformed to

$$a_0 \leftarrow a_1 \wedge \dots \wedge a_m, \neg L a_{m+1} \wedge \dots \wedge \neg L b_n$$

Such a transformation may distort the rule-like meaning of a clause when applied to extended programs [26]. The answer set of $\{b \leftarrow; a \leftarrow b\}$ is $\{a, b\}$, and the answer set of $\{b \leftarrow; \neg b \leftarrow \neg a\}$ is $\{b\}$ while the two programs are transformed to equivalent AE theories. So the transformation does not work well for extended programs.

It has been proposed by Lifschitz [40] and Chen [6] that a disjunctive extended clause

$$l_1, \dots, l_k \leftarrow l_{k+1}, \dots, l_m, \text{not} l_{m+1}, \dots, \text{not} l_{m+n}$$

is transformed into

$$(l_1 \wedge Ll_1) \vee \dots \vee (l_k \wedge Ll_k) \leftarrow (l_{k+1} \wedge Ll_{k+1}) \wedge \dots \wedge (l_m \wedge Ll_m) \wedge \neg Ll_{m+1} \wedge \dots \wedge \neg Ll_{m+n}.$$

For a program Π , Π^{ae} denotes the AE theory obtained from Π by the above transformation. Then the semantics for a disjunctive extended program Π is given by the stable expansions of the AE theories Π^{ae} .

It has also been proposed by Lifschitz [40] and Marek and Truszczyński [49] to transform an extended program to a reflexive autoepistemic theory. The semantics of reflexive autoepistemic theory is given by its *reflexive expansion*. Let Π be a set of sentences of an AE logic. A set E of sentences is a *reflexive expansion* of Π if

$$E = th(\Pi \cup \{\phi \equiv L\phi : \phi \in E\} \cup \{\neg L\phi : \phi \notin E\})$$

[81]. The difference between a reflexive expansion and a stable expansion is that positive introspection for a formula $\phi \in E$ is represented by $\phi \equiv L\phi$, rather than $L\phi$. The transformation of the above disjunctive extended program clause is

$$Ll_1 \vee \dots \vee Ll_k \leftarrow Ll_{k+1} \wedge \dots \wedge Ll_m \wedge L\neg Ll_{m+1} \wedge \dots \wedge L\neg Ll_{m+n}.$$

For a program Π , Π^{rae} denotes the AE theory obtained from Π by the above transformation. Then the semantics for a disjunctive extended program Π is given by the reflexive expansions of the AE theory Π^{rae} .

The following proposition can be extracted from [40, Main Theorem].

Proposition 5.4 *Let Π be a disjunctive extended program. For any consistent set S of literals, the following conditions are equivalent.*

- (a) S is a GL-answer set of Π ;
- (b) There exists a stable expansion E of Π^{ae} such that $S = E \cap Lit_L$;
- (c) There exists a reflexive expansion E of Π^{rae} such that $S = E \cap Lit_L$;

We have shown in Theorem 4.16 that every stable model of a normal program is an $\mathcal{AE}$ -preferable tuple of the program, and vice versa. The result can be extended to the GL-answer set semantics for extended programs with the above transformation. I.e. S is an answer set of Π iff H is $\mathcal{AE}$ -preferable of Π^{ae} iff H is RAE-preferable of Π^{rae} , where $H = \langle S, Lit_L - S \rangle$.

5.2 Argumentation Semantics

Dung extended the preferential extension semantics for normal programs to an argumentation semantics for extended programs [13]. In an extended logic program Π , literals of the form **not** l are referred to as *subjective* literals while ordinary literals are referred to as *objective* literals. An *argument* is defined as a set of subjective literals. A *proof* of an objective literal K is defined as a sequence $C_0, C_1, \dots, C_n$ of instantiated clauses of Π with the head of C_n being K such that for each i , $0 \leq i \leq n$, each objective literal l in the body of C_i is the head of C_j for some $j < i$. An objective literal K is *supported* by an argument A if there exists a proof $C_0, C_1, \dots, C_n$ of K such that all the subjective literals in the body of C_i , $0 \leq i \leq n$ are in A . An objective literal is supported by a set of arguments if it is supported by some argument in the set.

In the study of the preferential extension semantics, a negative literal $\neg a$ in a normal program is renamed as an abductable atom $\bar{a}$, which is considered as a possible hypothesis. The basic idea is that a hypothesis h is acceptable wrt a set of hypotheses if the set rejects all hypotheses that reject h . The idea for the argumentation semantics is that an argument A is acceptable wrt a set of arguments if the set 'attacks' all arguments that 'attack' A .

An argument A is *sound* if there does not exist an objective literal l such that A supports both l and $\neg l$. Let A and A' be arguments. A is said to *RAA-attack* (RAA for *Reductio Ad Absurdum*) A' if $A \cup A'$ is not sound. A is said to *g-attack* (g for ground) A' if there exists a subjective literal **not** $l \in A'$ such that A supports l . A set S of arguments is said to be *conflict-free* if no elements of S attack any elements in S . An argument A is *acceptable* wrt S if for any sound argument B , if B attacks A then B is g-attacked by some $A' \in S$.

Definition 5.2 A set S of arguments is *admissible* if

- (a) S is conflict-free; and
- (b) each argument in S is acceptable wrt S .

A *preferred extension* is a maximal (wrt set inclusion) admissible set of argument.

For an extended program Π and an argument A , Π^A denote the set obtained from $[\Pi]_{\mathbf{L}}$ by deleting each $\text{not}l \in A$ from the bodies of the clauses in $[\Pi]_{\mathbf{L}}$.

Lemma 5.5 *Let Π be an extended program and A an argument. For any objective literal l , A supports l if and only if $\Pi^A \vdash_{L_3} l$.*

Proof: The result follows from Definition 3.36 by structural induction. $\square$

Lemma 5.6 [13, Lemma 1] *Let an argument A be acceptable wrt an admissible set S of arguments. Then $S \cup \{A\}$ is admissible.*

Let S be a set of arguments. Then A_S denotes the argument $\bigcup_{A \in S} A$.

Lemma 5.7 *Let E be a preferred extension. Then for any nonempty subset S of E , $A_S \in E$.*

Proof: It suffices by Lemma 5.6 to show that A_S is acceptable wrt E . Let B attack A_S . If B g-attacks A_S , then B g-attacks some $A' \in S$. Because A' is acceptable wrt E , B is g-attacked by some $A'' \in E$. If B RAA-attacks A_S , $B \cup A_{S-\{A'\}}$ RAA-attacks some $A' \in S$ because $B \cup A_S = (B \cup A_{S-\{A'\}}) \cup A'$. Because A' is acceptable wrt E , $B \cup A_{S-\{A'\}}$ is g-attacked by some $A'' \in E$. Since A'' does not attack any arguments in S , A'' g-attacks B . Therefore, A_S is acceptable wrt E . $\square$

Proposition 5.8 *Let E be a nonempty preferred extension, and*

$$P = \{l : l \text{ is supported by } E\}, \text{ and } N = \{l : \text{not}l \in A_E\}.$$

Then $\langle P, N \rangle$ is L_3 -sound and L_3 -admissible.

Proof: By Lemma 5.7, $A_E \in E$. Let Π be an extended program. We show $H = \langle \emptyset, N \rangle$ is L_3 -admissible. If there exists $l \in N$ such that $H \uplus \Pi \vdash_{L_3} l$, then A_E supports l by Lemma 5.5. Thus A_E attacks some $A \in E$ since there exists $A \in E$ such that $\text{not}l \in A$. This contradicts that E is conflict-free. Therefore, H is L_3 -noncontradictory. For any $l \in N$ and any $H_l = \langle P_l, N_l \rangle$ such that $H_l \uplus \Pi \vdash_{L_3} l$, A_l supports l by Lemma 5.5, where $A_l = \{\text{not}l : l \in N_l\}$. Then A_l g-attacks A_E . Because A_E is acceptable wrt E , some $A' \in E$ g-attacks A_l , i.e. there exists $\text{not}l' \in A_l$ such that A' supports l' . Then $\langle \emptyset, N' \rangle \uplus \Pi \vdash_{L_3} l'$ by Lemma 5.5. Then $H \uplus \Pi \vdash_{L_3} l$ by Proposition 3.30. Therefore, l is L_3 -false-assumable wrt H , and H is L_3 -admissible. It follows from Lemma 5.5 and Proposition 3.30 that P is positively L_3 -founded wrt $\langle P, N \rangle$. Therefore, $\langle P, N \rangle$ is L_3 -admissible. Because A_E is sound, (otherwise it would RAA-attack all other $A \in E$), $\langle P, N \rangle$ is L_3 -sound by Lemma 5.5. $\square$

Theorem 5.9

- (a) For each preferred extension E , there exists an L_3 -preferable answer set S such that for any objective literal l , if l is supported by E , then $l \in S$.
- (b) There exists an L_3 -preferable answer set S such that every literal in S is not supported by any preferred extension.

Proof: Proof of (a):

When E is an empty set, the result is vacuously true. Suppose E is not an empty set. Let $\langle P, N \rangle$ be defined as in Proposition 5.8. By Theorem 3.20, there exists an L_3 -preferable tuple H' such that $\langle P, N \rangle$. It follows from Proposition 5.8 and Definition 3.41 that there exists a soundly L_3 -preferable tuple H' such that $\langle P, N \rangle \subseteq H'$. The result follows.

Proof of (b): The result follows from the following Example 5.1. $\square$

Example 5.1 Consider the program Π of Example 3.14.

$$\begin{array}{ll} \Pi : & a \leftarrow \text{not} p \\ & \neg a \leftarrow \text{not} q \\ & b \leftarrow \text{not} r \end{array}$$

Π has two soundly L_3 -preferable tuple $H_1 = \langle \{b, a\}, \{p\} \rangle$ and $H_2 = \langle \{b, \neg a\}, \{q\} \rangle$. Therefore, it has two L_3 -preferable answer sets $S_1 = \{b, a\}$ and $S_2 = \{b, \neg a\}$. The only preferred extension of Π is $E = \{\emptyset, \{\text{not} r\}\}$. Neither a nor $\neg a$ is supported by E .

The preferred extension semantics is intended to be a credulous semantics since a preferred extension is maximized (wrt set inclusion) to support as many literals as possible while preserving admissibility. However, Theorem 5.9 indicates that the preferred extension semantics is strictly less credulous than the L_3 -preferable answer set semantics.

5.3 New Semantics

The GL-answer set semantics represents an omniscient semantics and the preferred extension semantics represents a credulous semantics. Theorem 5.3, shows that the GL-answer set semantics is equivalent to the L_3 -stable answer set while Theorem 5.9 shows that L_3 -preferable answer set extends the preferred extension semantics. The L_m -well-founded answer set represents a skeptical semantics for extended programs

while the L_m -preferable answer set represent a credulous semantics. When $m = 2$, the semantics are capable of handling proof by cases. To the best of our knowledge, neither of them have been proposed before.

Kowalski and Sadri defined the ϵ -answer set semantics for programs with classical negation [31]. In [31], clauses with negative head are referred to as exceptions and exceptions have higher a priority than program clauses. So the potential contradiction in the answer set semantics is avoided by deleting clauses with head L if $\neg L$ is in an answer set. Poole and Goebel added both negation and disjunction to logic programming and studied the topic from the efficiency point of view [65].

Chapter 6

Classification of Semantics

6.1 Introduction

The classical first order logic is inadequate to formalize human reasoning [51, 60]. The difficulty is primarily due to the common sense and nonmonotonic nature of human reasoning. In order to find formal foundations of nonmonotonic reasoning, researchers have proposed several formalizations of nonmonotonic reasoning, among which the best known are McCarthy's circumscription [52, 53], Reiter's default theory [76], Moore's autoepistemic logic [61] which is a reconstruction of McDermott and Doyle's nonmonotonic logic [55, 56], Reiter's closed world assumption [75] and its extension, Minker's generalized closed world assumption (GCWA) [58]. There is no doubt that the area of nonmonotonic reasoning is of crucial and growing importance to the whole field of AI [27].

Since none of the formal systems of nonmonotonic reasoning have the property — monotonicity, research has been done to postulate some desirable properties of logics for nonmonotonic reasoning [44, 32]. The approach, probably first suggested by Gabbay [19], was to focus the study of nonmonotonic logics on their consequence relations. The fundamental question Gabbay asked was "What are the minimal conditions a consequence relation should satisfy to represent a bona fide nonmonotonic logic?" He proposed three conditions: *reflexivity*, *cut* and *cautious monotonicity* (see equations in next section). Cautious monotonicity was called *weak monotonicity* in [19]. Cut and Cautious monotonicity together amount to the very important property — cumulativity.

The idea to study the deduction process in terms of the consequence relations dates back to Tarski [87] and Gentzen [85]. Tarski studied the consequences of arbitrary sets of formulas in terms of consequence operator Cn whereas Gentzen restricted himself to finite sets and studied the consequence relation $\vdash$ although the difference

between the two approaches is small for monotonic consequence relation in the presence of compactness. Makinson developed an infinitistic approach for nonmonotonic consequence operators in the style of Tarski [44], and Kraus, Lehmann and Magidor studied various nonmonotonic consequence relations in the style of Gentzen [32]. The two approaches are closely related. One of the most important properties both approaches identify is cumulativity. We will review very briefly both of them in this chapter.

The close relationship of logic programming and nonmonotonic reasoning has been established in [24, 68, 70, 72, 12, 50, 94, 96] and in Chapter 3. Most of the work is on expressions of various semantics for logic programming in terms of different formalisms for nonmonotonic reasoning, or on elucidation of the coincidence of various semantics for logic programming with the semantics of different formalisms for nonmonotonic reasoning. All the semantics of logic programs proposed so far are not monotonic in the sense that answers which used to be correct may not be so in light of new information or facts. As a quite natural outcome of Makinson's and Kraus, Lehmann and Magidor's work, semantics for logic programming have been analyzed against those desirable properties of nonmonotonic logics [8, 10, 36]. Furthermore, various desirable properties have also been postulated specifically for semantics of logic programming [28, 10, 9]. Such properties include *locality*, *logical implication*, *GCWA* [28], *relevance and modularity* [10, 9], etc. In Section 6.2, we will enhance the principle of locality, first proposed by us in [28] and referred to as the principle of relevance by Dix in [10]. In Section 6.3, we will analyze semantics for logic programming represented in our framework against those desirable properties of nonmonotonic logics. In the course of presentation, we will also compare our work with Dix's work [8, 10, 9].

6.2 The Principle of Locality

6.2.1 Relevant Set and Relevant Program

Two of the major tasks in searching for an appropriate declarative semantics for logic programming are to define the meaning of the negation **not** in a program, and to define the meaning of the implication $\leftarrow$ in a program. In our framework, the negations are taken as acceptable hypotheses — ones in an admissible hypothesis set, and the meaning of the implication symbol is characterized by the incorporating operator $\oplus$ and oriented consequence operator $\vdash_{L_m}$. It is clear that $\leftarrow$ in logic programming is not to be taken as material implication. A clause in implication form may be considered as a *definition* of the head of the program clause. Therefore, the truth value of

any atom should be determined only by those clauses that may be used to derive the atom. This motivates the following definitions of *relevant sets* and *relevant programs*. First we define the dependence relation on the Herbrand base B_L of a language L in which programs are written.

Definition 6.1 Let Π be a program. The binary relation $\leq_\Pi$ on B_L is defined as the smallest, reflexive and transitive relation satisfying that $a \leq_\Pi b$ if there exists an instantiated clause α of Π such that a appears in the head and b appears in the body.

If a appears in the head of a clause and b appears in the body of the clause, the meaning or truth value of a depends on the meaning or truth value of b . We would simply say a depends on b . Thus $a \leq_\Pi b$ means that the meaning of a hereditarily depends on the meaning of b . Przymusiński introduced a similar relation $\leq$ in [69] for normal programs. The only difference is that his relation is not required to be reflexive, and this difference is not essential. $\leq$ (together with another relation $<$) is used to define stratifications or to rank the priorities of atoms in the Herbrand base. Clearly, the same atom has the same priority. We will use the relation $\leq_\Pi$ to express the relevance of the meanings of atoms. The same atom is vacuously relevant with itself.

Example 6.1 Consider the program Π .

$$\begin{aligned} b &\leftarrow \text{nota} \\ a &\leftarrow \text{not}b \\ p &\leftarrow a \end{aligned}$$

We have $a \leq_\Pi b$, $b \leq_\Pi a$, $p \leq_\Pi a$ and $p \leq_\Pi b$. Thus a and b depend on each other and they are relevant to each other. On the other hand, neither a nor b depends on p , and p is irrelevant to the truth values of a and b . Since $p \leftarrow a$ is considered a definition of p , it should not affect the meaning of either a or b . In other words, the meaning of a and b should be determined only by $a \leftarrow \text{not}b$ and $b \leftarrow \text{nota}$. Of course, the meaning of p should be determined by the whole program since it depends on every atom and every atom is relevant to p .

Now we are in the position to define the relevant set and the relevant program of an atom.

Definition 6.2 Let Π be a program and a an atom. The *relevant set* $R(a)$ of a wrt Π is defined as

$$R_\Pi(a) = \{b : a \leq_\Pi b\}$$

The *relevant program* $\Pi(a)$ of a wrt Π is defined as

$$\Pi(a) = \{l \leftarrow l_1, \dots, l_n \in [\Pi]_L : b \in R_\Pi(a) \text{ appears in } l\}.$$

The relevant set $R_{\Pi}(a)$ of a contains all and only atoms on which a depends. These atoms are relevant to a and appear in the relevant program of a . Actually, these are the only atoms that may appear in the relevant program. (Note that when $\Pi(a) = \emptyset$, $R_{\Pi}(a) = \{a\}$.) The relevant program $\Pi(a)$ of a is the set of all instantiated clauses $b \leftarrow l_1, \dots, l_n$ in Π such that a hereditarily depends on b with respect to Π . These clauses are the relevant clauses that may be used to derive a and should uniquely determine the truth value of a .

A concept of *the determinants set of an atom*, which is similar to the relevant set of an atom, and the concept of relevant program of an atom were first introduced in [28]. The determinants and the relevant program of an atom introduced in [28] are determined dynamically with respect to certain semantics. The relevant set and the relevant program introduced in Definition 6.2 are determined statically by the dependency among atoms specified by the original program. They are a simplified form of determinant sets and relevant programs dynamically determined. The same definition as Definition 6.2 was also presented independently by Dix in [10].

6.2.2 Principle of Locality

Given a program, a semantics for logic programming specifies the meaning or truth value of every atom. According to the discussion in the last section, the meaning of an atom should be determined locally by its relevant program instead of the whole program. Then under the same semantics, the whole program and the relevant program of an atom should specify the same truth value for the atom. We introduce the following notations to formalize this principle. Let T , F , and R be sets. Then $\langle T, F \rangle \cap R$ denotes $\langle T \cap R, F \cap R \rangle$. Clearly, such a notation is readily extended to a set of tuples. For a set S of tuples, $S \cap R = \{I \cap R : I \in S\}$. We assume the semantics SEM is represented as a set of partial interpretations. Clearly, all semantics given in Chapter 4 are already of this form. The answer set semantics can naturally be transformed into this form as follows. An answer set S can be transformed into $\langle \{a : a \in S\}, \{b : \neg b \in S\} \rangle$.

Definition 6.3 A semantics SEM is said to obey the *Principle of Locality* if for any program Π and any ground atom a ,

$$SEM(\Pi) \cap R_{\Pi}(a) = SEM(\Pi(a)) \cap R_{\Pi}(a),$$

where $R_{\Pi}(a)$ and $\Pi(a)$ are the relevant set and program of a respectively.

In terms of partial interpretations, $SEM(\Pi)$ specifies all the possible worlds described by the program Π while $SEM(\Pi(a))$ specifies all the possible worlds described

by $\Pi(a)$. $R_\Pi(a)$ can be considered all the propositions in a local world of a . The principle of locality says that Π and $\Pi(a)$ should specify the same possible local worlds of a . The principle of locality implies that the truth value of any atom a is uniquely determined by its relevant program because $a \in R_\Pi(a)$. We strongly believe that a proper declarative semantics for logic programming should obey the principle of locality.

For any atom b in the relevant set $R_\Pi(a)$ of a , the relevant set and program of b are subsets of the relevant set and program of a respectively, i.e. $R_\Pi(b) \subseteq R_\Pi(a)$ and $\Pi(b) \subseteq \Pi(a)$ because $\leq_\Pi$ is transitive. Furthermore, the meaning of b should also be determined locally by $\Pi(b)$. The principle of locality also says that $\Pi(a)$ and $\Pi(b)$ should specify the same truth value for b under any semantics satisfying the principle.

A similar principle was also proposed in [28, 10] for normal programs. (It was called the principle of relevance in [10].) The principle only requires that the truth value of an atom a be the same in the semantics of the whole program and its relevant program, but there are no requirements on the truth value of the rest of the relevant set.

In the following proposition and lemmas, Π is a program. $R(a) \subseteq \delta_\Pi$ denotes the set of all relevant hypothesis, i.e.

$$R(a) = \{l \in \delta_\Pi : l \in R_\Pi(a) \text{ or } l = \neg b \text{ such that } b \in R_\Pi(a)\}$$

where $R_\Pi(a)$ is the relevant set of a wrt Π . $\Pi(a)$ is the relevant program of an atom a wrt Π . $\bar{R}(a)$ denote the set of irrelevant hypothesis, i.e. $\bar{R}(a) = \delta_\Pi - R(a)$. L_m denotes Lukasiewiczian m -valued logic.

Proposition 6.1 *Both L_m -well-founded and L_m -preferable semantics obey the principle of locality. The $\mathcal{AE}$ -preferable semantics does not.*

Proof: It suffices to show that for every soundly L_m -saturated tuple $H = \langle P, N \rangle$ of Π , there exists a soundly L_m -saturated (L_m -preferable) tuple $H_a = \langle P_a, N_a \rangle$ of $\Pi(a)$ such that $H_a \cap R(a) = H \cap R(a)$, and vice versa.

It follows from Lemma 6.2 that any L_m -sound and L_m -admissible tuple I of Π , $I \cap R(a)$ is L_m -sound and L_m -admissible of $\Pi(a)$, and vice versa. Then the result follows from Lemma 6.3 and 6.4.

For the $\mathcal{AE}$ -preferable semantics, the result is shown by Example 6.2. $\square$

We now proceed to prove the following lemmas.

Lemma 6.2 *For any clause $\gamma : l \leftarrow l_1, \dots, l_n$ with $l \in R(a)$ and $H \subseteq \Delta_\Pi$, $(H \cap R(a)) \uplus \Pi(a) \vdash_{L_m} \gamma$ if and only if $H \uplus \Pi \vdash_{L_m} \gamma$.*

Proof: The result follows immediately from the Definition 3.36 and 6.2 by structural induction. $\square$

Lemma 6.3 *For every L_m -saturated (L_m -preferable) tuple $H = \langle P, N \rangle$ of Π , there exists an L_m -saturated (L_m -preferable) tuple $H_a = \langle P_a, N_a \rangle$ of $\Pi(a)$ such that $H_a \cap R(a) = H \cap R(a)$.*

Proof: Let $P_a = P \cap R(a)$, $N_a = N \cup \bar{R}(a)$ and $H_a = \langle P_a, N_a \rangle$. Then $H_a \cap R(a) = H \cap R(a)$. It follows from Proposition 3.35 and Lemma 6.2 that H_a is an L_m -saturated model of $\Pi(a)$ because H is L_m -saturated of Π .

We now prove that H_a is L_m -preferable of $\Pi(a)$ if H is L_m -preferable of Π . Let $N = \langle P_N, N_N \rangle$ be an L_m -preferable model of $\Pi(a)$ such that $H_a \subseteq N$. Note that $P_a \subseteq R(a)$, $\bar{R}(a) \subseteq N_a$, $P_N \subseteq R(a)$ and $\bar{R}(a) \subseteq N_N$ because no literal in $\bar{R}(a)$ appears in $\Pi(a)$. Let $P' = P \cup P_N$, $N' = N \cup (N_N \cap R(a))$ and $H' = \langle P', N' \rangle$. By Lemma 6.2, H' satisfies condition (a), (b) and (c) of Proposition 3.35. Therefore, H' is L_m -saturated of Π . Since H is L_m -preferable and $H \subseteq H'$, $H = H'$. Then $N \subseteq H_a$ because $P_N \subseteq P' \cap R(a) = P \cap R(a) = P_a$ and $N_N \subseteq N' \cup \bar{R}(a) = N \cup \bar{R}(a) = N_a$. We proved that H_a is L_m -preferable of $\Pi(a)$ since $H_a = N$. $\square$

Lemma 6.4 *For every L_m -saturated (L_m -preferable) tuple $H_a = \langle P_a, N_a \rangle$ of $\Pi(a)$, there exists an L_m -saturated (L_m -preferable) tuple $H = \langle P, N \rangle$ of Π such that $H \cap R(a) = H_a \cap R(a)$.*

Proof: Let $H' = H_a \cap R(a)$. We prove that H' is $\mathcal{L}$ -admissible of Π . Consider $b \in N_a \cap R(a)$. If $H' \cup \Pi \vdash_{L_m} b$, then $H' \cup \Pi(a) \vdash_{L_m} b$ by Lemma 6.2. Thus $H_a \cup \Pi(a) \vdash_{L_m} b$ by Proposition 3.30. Then H_a is not L_m -noncontradictory of $\Pi(a)$. Contradiction. Therefore, H' is L_m -noncontradictory of Π . For any set $H_b = \langle P_b, N_b \rangle$ such that $H_b \cup \Pi \vdash_{L_m} b$, $(H_b \cap R(a)) \cup \Pi(a) \vdash_{L_m} b$ by Lemma 6.2 and $b \in R(a)$. There exists $b' \in N_b \cap R(a)$ such that $H_a \cup \Pi(a) \vdash_{L_m} b'$ because $b \in N_a$ is L_m -false-assumable of $\Pi(a)$ wrt H_a . Thus, $H' \cup \Pi \vdash_{L_m} b'$ by Lemma 6.2. Then, b is L_m -false-assumable of Π wrt H' . It follows from Lemma 6.2 that H' is positively L_m -founded. Hence, H' is $\mathcal{L}$ -admissible of Π .

Let H^* be the L_m -saturated closure of H' . By Proposition 3.24, H^* is the iterated fixed point of Ψ_Π defined in Definition 3.28 starting from $H_0 = H'$. We show $H^* \cap R(a) = H_a \cap R(a)$ by transfinite induction that for any ordinal α , $H_\alpha \cap R(a) = H_a \cap R(a)$, where $H_\alpha = \langle P_\alpha, N_\alpha \rangle$.

$H_0 \cap R(a) = H_a \cap R(a)$ by definition.

Assume $H_\beta \cap R(a) = H_a \cap R(a)$, for all $\beta < \alpha$. If $\alpha \neq 0$ is a limit, then clearly $H_\alpha \cap R(a) = H_a \cap R(a)$. If α is not, $\alpha - 1$ exists. Let $b \in R(a)$ be L_m -false-assumable of Π wrt $H_{\alpha-1}$. For any set $H_b = \langle P_b, N_b \rangle$ such that $H_b \cup \Pi(a) \vdash_{L_m} b$,

$H_b \cap R(a) \uplus \Pi(a) \vdash_{L_m} b$. By Lemma 6.2, $H_b \cap R(a) \uplus \Pi \vdash_{L_m} b$. There exists $b' \in N_b \cap R(a)$ such that $H_{\alpha-1} \uplus \Pi \vdash_{L_m} b'$. Thus $H_{\alpha-1} \cap R(a) \uplus \Pi(a) \vdash_{L_m} b'$ by Lemma 6.2. It follows $H_a \uplus \Pi(a) \vdash_{L_m} b'$ because $H_{\alpha-1} \cap R(a) = H_a \cap R(a)$. Therefore, b is L_m -false-assumable of $\Pi(a)$ wrt H_a . Then $b \in N_a$ because N_a is L_m -saturated of $\Pi(a)$. It follows from Lemma 6.2 that $H_\alpha \cap R(a) \subseteq H_a \cap R(a)$. Since $H_{\alpha-1} \cap R(a) \subseteq H_\alpha \cap R(a)$ and $H_{\alpha-1} \cap R(a) = H_a \cap R(a)$, $H_\alpha \cap R(a) = H_a \cap R(a)$. And the induction is completed.

We then show that there exists an L_m -preferable model N of Π such that $N \cap R(a) = H_a \cap R(a)$ if H_a is L_m -preferable of $\Pi(a)$. Suppose H_a is L_m -preferable of $\Pi(a)$. Let $N = \langle P_N, N_N \rangle$ be an L_m -preferable model of Π such that $H^* \subseteq N$. (N exists by Theorem 3.20.) Thus $H_a \cap R(a) = H^* \cap R(a) \subseteq N \cap R(a)$. It follows that $H_a \subseteq \langle P_N \cap R(a), N_N \cup \bar{R}(a) \rangle$ since $P_a \subseteq R(a)$, $\bar{R}(a) \subseteq N_a$ and $\bar{R}(a) \subseteq N_N$. In the proof of Lemma 6.3, we have shown that $\langle P_N \cap R(a), N_N \cup \bar{R}(a) \rangle$ is L_m -preferable of $\Pi(a)$. Because H_a is L_m -preferable of $\Pi(a)$, $H_a = \langle P_N \cap R(a), N_N \cup \bar{R}(a) \rangle$. Therefore, $N \cap R(a) = H_a \cap R(a)$. $\square$

Example 6.2 Consider the following program Π' obtained from the program Π in Example 6.1 by adding one more clause $p \leftarrow \text{not} p$.

$$\begin{aligned} b &\leftarrow \text{not} a \\ a &\leftarrow \text{not} b \\ p &\leftarrow \text{not} p \\ p &\leftarrow a \end{aligned}$$

$M_1 = \langle \{a, p\}, \{b\} \rangle$ is the only stable model of Π' and $SEM_{\mathcal{AE}-p}(\Pi') = M_1$. (Note that $\mathcal{AE}$ -preferable semantics coincides with the stable model semantics.) However, the relevant set and program of a and b are $\{a, b\}$ and $\Pi'' = \{b \leftarrow \text{not} a, a \leftarrow \text{not} b\}$. Π'' has two stable models $\langle \{a\}, \{b, p\} \rangle$ and $\langle \{b\}, \{a, p\} \rangle$. Clearly, $SEM_{\mathcal{AE}-p}(\Pi') \cap \{a, b\} \neq SEM_{\mathcal{AE}-p}(\Pi'') \cap \{a, b\}$.

Both $M_1 = \langle \{a, p\}, \{b\} \rangle$ and $M_2 = \langle \{b\}, \{a, p\} \rangle$ are stable models of Π . Then under the $\mathcal{AE}$ -preferable semantics, we can not infer p, a and $\text{not} b$ from Π . $M_1 = \langle \{a, p\}, \{b\} \rangle$ is the only stable model of Π' and $SEM_{\mathcal{AE}-p}(\Pi') = M_1$. We can infer p, a and $\text{not} b$ from Π' under the semantics. In other words, adding the clause $p \leftarrow \text{not} p$ changes the meaning of every atom even if neither a nor b depend on p and their relevant programs wrt both Π and Π' are the same program Π'' . Such behavior of $\mathcal{AE}$ -preferable, i.e. stable model semantics is against the principle of locality.

6.2.3 Principle of Setwise Locality

This section introduces a stronger principle of locality. The principle of locality in Definition 6.3 says that the meaning of a single atom is to be specified by the relevant program of the single atom. Such a principle can be naturally extended to a set of atoms. The *relevant set* $R_{\Pi}(S)$ of a set S of ground atoms wrt a program Π is defined as

$$R_{\Pi}(S) = \{b : a \leq_{\Pi} b \text{ for some } a \in S\}.$$

The *relevant program* $\Pi(S)$ of S wrt Π is defined the same as before, i.e.

$$\Pi(S) = \{l \leftarrow l_1, \dots, l_n \in [\Pi]_{\mathbf{L}} : b \in R_{\Pi}(S) \text{ appears in } l\}.$$

Naturally, we have the following definition for the principle of setwise locality.

Definition 6.4 A semantics SEM is said to obey the *Principle of Setwise Locality* if for any program Π and any set S of ground atoms,

$$SEM(\Pi) \cap R_{\Pi}(S) = SEM(\Pi(S)) \cap R_{\Pi}(S).$$

The principle of locality in Definition 6.3 may be called a principle of pointwise locality in view of the above definition. The intuition behind this setwise principle is the same as the pointwise principle. $R_{\Pi}(S)$ can be considered all the propositions in a local world of the set S . The setwise principle says that the semantics of a whole program and the relevant program of a set S should specify the same possible local worlds for S . The principle of setwise locality is strictly stronger than the principle of locality. When the set S is singleton, the setwise principle reduces to the pointwise principle. And one can easily find a semantics which obeys the principle of pointwise locality but does not obey the principle of setwise locality. We also believe that the declarative semantics for logic programming should obey the principle of setwise locality.

Proposition 6.5 Both L_m -well-founded and L_m -preferable semantics obey the principle of setwise locality.

Proof: In Lemma 6.2, 6.3, 6.4 and their proofs, replace $R(a)$ with $R_{\Pi}(S)$, $\Pi(a)$ with $\Pi(S)$. The proofs are still correct and the lemmas still hold for any set S of ground atoms. Then the result follows immediately. $\square$

Obviously, the $\mathcal{AE}$ -preferable semantics does not obey the setwise principle since it does not obey the pointwise one by Proposition 6.1.

6.2.4 Applications of the Principles of Locality

The least Herbrand model semantics provides a satisfactory declarative reading for definite programs [3, 14]. It can also be characterized as the least fixed point of the operator T_{Π} . One can find out that all the semantics proposed and discussed in the preceding chapters coincide with the least Herbrand semantics when restricted to definite programs. It seems fair to say that none of the semantics proposed for normal programs, which allow negations, is as satisfactory as the least Herbrand model semantics is for definite programs. Another satisfactory semantics for programs with negations is the *standard model* semantics proposed by Apt, Blair, and Walker [2] and independently by Van Gelder [22] for *stratified programs*. A stratified program allows both negation and recursion, but not recursion through negation. The standard model semantics extends the least fixed point semantics, and is defined as an iterated fixed point of the operator T_{Π} , which, in general, is nonmonotonic for program with negation. In this section, we will apply the principle of setwise locality to stratified programs and define the standard model in terms of the least Herbrand model.

Let Π be a program. By a *definition* of a predicate symbol p we mean the subset of Π consisting of all program clauses with p appearing in the heads. We say p occurs *positively* in a positive literal and *negatively* in a negative literal.

Definition 6.5 [2, Definition 3] A program Π is called *stratified* if there is a partition

$$\Pi = \Pi_1 \cup \dots \cup \Pi_n$$

such that the following two conditions holds for $i = 1, \dots, n$:

1. if a predicate symbol occurs positively in a clause in Π_i , then its definition is contained within $\cup_{j \leq i} \Pi_j$.
2. if a predicate symbol occurs negatively in a clause in Π_i , then its definition is contained within $\cup_{j < i} \Pi_j$.

Π_1 can be empty. We say that Π is *stratified* by $\Pi_1 \cup \dots \cup \Pi_n$ and each Π_i is called a *stratum* of Π . We let $\bar{\Pi}_i$ denote $\Pi_1 \cup \dots \cup \Pi_i$. Thus $\bar{\Pi}_n = \Pi$.

Clearly, any predicate symbol p occurring negatively in Π_i does not occur in the head of any clause in Π_j with $j \geq i$.

Definition 6.6 Let a program Π in a language $\mathbf{L}$ be stratified by $\Pi_1 \cup \dots \cup \Pi_n$. We put

$$\begin{aligned} P_n &= B_{\mathbf{L}}; \\ P_i &= P_{i+1} - \{a \in B_{\mathbf{L}} : \text{the predicate symbol of } a \text{ occurs in the head of a clause in } \Pi_{i+1}\}, \\ 1 &\leq i < n. \end{aligned}$$

Proposition 6.6 *Let a program Π be stratified by $\Pi_1 \cup \dots \cup \Pi_n$. Then*

$$R_{\Pi}(P_i) = P_i \text{ and } \Pi(P_i) = [\bar{\Pi}_i]_{\mathbf{L}},$$

where $1 \leq i \leq n$ and $[\bar{\Pi}_i]_{\mathbf{L}}$ is the Herbrand instantiation of $\bar{\Pi}_i$.

Proof: It follows from definitions that $R_{\Pi}(P_n) = P_n$ and $\Pi(P_n) = [\bar{\Pi}_n]_{\mathbf{L}}$. Let $i < n$. $P_i \subseteq R_{\Pi}(P_i)$ because $\leq_{\Pi}$ is reflexive. Let $j > i$. A predicate symbol occurring in a head of a clause in Π_j does not occur in $\bar{\Pi}_i$ by Definition 6.5, and the ground atoms of the predicate are not in P_i by Definition 6.6. Thus $R_{\Pi}(P_i) \subseteq P_i$. Hence $R_{\Pi}(P_i) = P_i$. For any $\alpha : a \leftarrow l_1, \dots, l_m$ and $\alpha \in [\Pi]_{\mathbf{L}}$, if $a \in P_i$, then $\alpha \in [\bar{\Pi}_i]_{\mathbf{L}}$ because the predicate of a does not occur in the heads of any clause in Π_j by Definition 6.6. Thus $\Pi(P_i) \subseteq [\bar{\Pi}_i]_{\mathbf{L}}$. If $\alpha \in [\bar{\Pi}_i]_{\mathbf{L}}$, then the predicate of a does not occur in Π_j by Definition 6.5. Thus $a \in P_i$. Therefore, $[\bar{\Pi}_i]_{\mathbf{L}} \subseteq \Pi(P_i)$. Hence $\Pi(P_i) = [\bar{\Pi}_i]_{\mathbf{L}}$. $\square$

By the principle of setwise locality, the meanings of atoms in P_1 are to be determined by the relevant program $\Pi(P_1)$ of P_1 . By Proposition 6.6, $\Pi(P_1)$ is $[\bar{\Pi}_1]_{\mathbf{L}}$, i.e. $[\Pi_1]_{\mathbf{L}}$, which is the ground representation of Π_1 . Therefore, the meaning of atoms in P_1 are to be determined by Π_1 . Since Π_1 is the first stratum of $\Pi_1 \cup \dots \cup \Pi_n$, no negation appears in Π_1 by Definition 6.5. Therefore, Π_1 is a definite program. According to the least Herbrand model semantics, the declarative reading of Π_1 is specified by the least Herbrand model of Π_1 . Consider P_2 . The meaning of atoms in P_2 are to be determined by $\Pi(P_2) = [\bar{\Pi}_2]_{\mathbf{L}}$. Any predicate symbol p occurring negatively in Π_2 does not occur in the head of any clause in Π_j with $j \geq 2$. Therefore, the ground atoms formed from p are in P_1 , and the meanings of them are to be determined by the least Herbrand model of Π_1 . Let N_1 be the least Herbrand model of Π_1 . (N_1 is of the form of a set of ground atoms.) Then for any instantiated clause $\alpha : a \leftarrow a_1, \dots, a_k, \neg b_1, \dots, \neg b_m$ of $\bar{\Pi}_2$, α is useless if there exists i such that $b_i \in N_1$, and $\neg b_j$ may be deleted from α if $b_j \notin N_1$ because the meaning of b_i has been determined to be false by N_1 . Recall Definition 4.7 for GL-transformation. Thus $\frac{\bar{\Pi}_2}{N_1} \cup N_1$ is the program obtained from $\bar{\Pi}_2$ incorporated with N_1 , where $\frac{\bar{\Pi}_2}{N_1}$ is the GL-transformation of $\bar{\Pi}_2$ modulo N_1 . Since Π_1 is a subset of $\bar{\Pi}_2$ and N_1 is the set of logical consequences of Π_1 by Theorem 4.18, we can consider $\frac{\bar{\Pi}_2}{N_1}$ only without losing any information expressed by N_1 . As $\frac{\bar{\Pi}_2}{N_1}$ contains no negation, the declarative reading of it is also to be specified by the least Herbrand model of it. Such a discussion can also be applied to P_3, P_4 , and so on, till P_n . This lead us to the following definition.

Definition 6.7 *Let a program Π be stratified by $\Pi_1 \cup \dots \cup \Pi_n$.*

$$N_0 = \emptyset;$$

$$N_1 = \text{the least Herbrand model of } \frac{\bar{\Pi}_1}{N_0};$$

$N_2 =$ the least Herbrand model of $\frac{\bar{\Pi}_2}{N_1}$;

...

$N_n =$ the least Herbrand model of $\frac{\bar{\Pi}_n}{N_{n-1}}$.

Lemma 6.7 $N_1 \subseteq N_2 \subseteq \dots \subseteq N_n$ and $\frac{\bar{\Pi}_i}{N_{i-1}} = \frac{\bar{\Pi}_i}{N_i}$ with $1 \leq i \leq n$.

Proof: By Theorem 4.20, $N_i = T_{\frac{\bar{\Pi}_i}{N_{i-1}}} \uparrow^\omega$ for $i, 1 \leq i \leq n$. We prove by induction that $N_{i-1} \subseteq N_i$ and $\frac{\bar{\Pi}_i}{N_{i-1}} = \frac{\bar{\Pi}_i}{N_i}$ for each i satisfying $1 \leq i \leq n$. $N_0 \subseteq N_1$ since $N_0 = \emptyset$. $\frac{\bar{\Pi}_1}{N_0} = \frac{\bar{\Pi}_1}{N_1}$ because $\bar{\Pi}_1$ is Π_1 which is definite. Assume $N_{k-1} \subseteq N_k$ and $\frac{\bar{\Pi}_k}{N_{k-1}} = \frac{\bar{\Pi}_k}{N_k}$ for $1 \leq k < n$. Since $\frac{\bar{\Pi}_{k+1}}{N_k} = \frac{\bar{\Pi}_{k+1}}{N_k} \cup \frac{\bar{\Pi}_k}{N_k}$, $T_{\frac{\bar{\Pi}_{k+1}}{N_k}} \uparrow^\omega \subseteq T_{\frac{\bar{\Pi}_{k+1}}{N_k} \cup \frac{\bar{\Pi}_k}{N_k}} \uparrow^\omega = N_{k+1}$. Since $\frac{\bar{\Pi}_k}{N_{k-1}} = \frac{\bar{\Pi}_k}{N_k}$, $N_k = T_{\frac{\bar{\Pi}_k}{N_k}} \uparrow^\omega$. So $N_k \subseteq N_{k+1}$. For each $a \in N_{k+1} - N_k$, the predicate symbol p occurring in a occurs in the head of a program clause in Π_{k+1} . (Otherwise, $a \in T_{\frac{\bar{\Pi}_k}{N_k}} \uparrow^\omega$, and $a \in N_k$.) Thus p does not occur negatively in $\bar{\Pi}_{k+1}$ by Definition 6.5. Then $\frac{\bar{\Pi}_{k+1}}{N_k} = \frac{\bar{\Pi}_{k+1}}{N_{k+1}}$. $\square$

From the above discussion, the semantics of a program Π stratified by $\Pi_1 \cup \dots \cup \Pi_n$ is to be specified by N_n according to the least Herbrand model semantics and the principle of setwise locality. In the following, we will show that N_n is the standard model of Π . It has been shown that the standard model is independent of the stratification [2, Theorem 11]. Thus the declarative reading of a stratified program is to be given by N_n .

The operator T_Π defined in Definition 4.12 is nonmonotonic in general for a normal program Π . The ordinal power defined for monotonic mapping in Definition 3.22 is not applicable to T_Π . The ordinal power of a nonmonotonic mapping Φ is defined as follows [2, p 100].

$$\Phi \uparrow 0(x) = x,$$

$$\Phi \uparrow (n+1)(x) = \Phi(\Phi \uparrow n(x)) \cup \Phi \uparrow n(x),$$

$$\Phi \uparrow \omega(x) = \text{lub}(\{\Phi \uparrow n(x) : n < \omega\}).$$

Ordinal powers beyond ω may be defined accordingly, but are not needed here. $\Phi \uparrow n(x)$ should not be confused with the ordinal power $\Phi \uparrow^n(x)$ of a monotonic mapping Φ defined in Proposition 3.17. An obvious proof by induction shows that when Φ is monotonic, both definitions lead to the same value of $\Phi \uparrow n(\perp)$ and $\Phi \uparrow^n(\perp)$. (When Φ is T_Π , $\perp$ is $\emptyset$.) For $x \neq \perp$, $\Phi \uparrow n(x)$ and $\Phi \uparrow^n(x)$ are the same only when $x \leq \Phi(x)$ ($x \subseteq \Phi(x)$ if Φ is T_Π). Clearly

$$T_\Pi \uparrow \omega(I) = \cup_{n=0}^{\infty} T_\Pi \uparrow n(I).$$

for any subset $I \subseteq B_L$.

Let a program Π be stratified by $\Pi_1 \cup \dots \cup \Pi_n$. We put

$$M_1 = T_{\Pi_1} \uparrow \omega(\emptyset),$$

$$M_2 = T_{\Pi_2} \uparrow \omega(M_1),$$

...

$$M_n = T_{\Pi_n} \uparrow \omega(M_{n-1}),$$

Then the standard model M of the stratified program Π is defined as M_n [2, p 108]. It has been shown that M is independent of the stratification [2, Theorem 11].

Proposition 6.8 *Let a program Π be stratified by $\Pi_1 \cup \dots \cup \Pi_n$. Then $N_i = M_i$ for all i , $1 \leq i \leq n$, where N_i is defined in Definition 6.7.*

Proof: By Theorem 4.20, $N_i = T_{\frac{\bar{n}_i}{N_{i-1}}} \uparrow \omega$ for i , $1 \leq i \leq n$. Since Π_1 is definite, T_{Π_1} is monotonic. Since $\frac{\bar{n}_1}{N_0}$ is $[\Pi_1]_L$, T_{Π_1} is the same as $T_{\frac{\bar{n}_1}{N_0}}$. Thus $T_{\Pi_1} \uparrow \omega(\emptyset) = T_{\frac{\bar{n}_1}{N_0}} \uparrow \omega$ by an obvious proof of induction. So $N_1 = M_1$. Assume $N_k = M_k$ for $1 \leq k < n$.

Let $M_{k+1,j} = T_{\Pi_{k+1}} \uparrow j(M_k)$. We prove by induction that $M_{k+1,j} \subseteq N_{k+1}$ for each j . $M_{k+1,0} = M_k = N_k$. By Lemma 6.7, $N_k \subseteq N_{k+1}$. Therefore, $M_{k+1,0} \subseteq N_{k+1}$. Assume $M_{k+1,j} \subseteq N_{k+1}$ for some $j \geq 0$. For any $a \in M_{k+1,j+1} - M_{k+1,j}$, $a \in T_{\Pi_{k+1}}(M_{k+1,j}) - M_{k+1,j}$. There exists an instantiated clause $\alpha : a \leftarrow a_1, \dots, a_l, \neg b_1, \dots, \neg b_m$ of Π_{k+1} such that each $a_i \in M_{k+1,j}$ and each $b_i \notin M_{k+1,j}$. Since $M_k \subseteq M_{k+1,j}$ and $N_k = M_k$, each $b_i \notin N_k$. Thus $a \leftarrow a_1, \dots, a_l$ is in $\frac{\bar{n}_{k+1}}{N_k}$. Since each $a_i \in M_{k+1,j}$ and $M_{k+1,j} \subseteq N_{k+1}$, each $a_i \in N_{k+1}$. Because N_{k+1} is a fixed point of $T_{\frac{\bar{n}_{k+1}}{N_k}}$, $a \in N_{k+1}$. So $M_{k+1,j+1} \subseteq N_{k+1}$. Induction is completed. Since $M_{k+1} = T_{\Pi_{k+1}} \uparrow \omega(M_k) = \bigcup_{j=0}^{\infty} M_{k+1,j}$, we have proved that $M_{k+1} \subseteq N_{k+1}$.

Let $N_{k+1,j} = T_{\frac{\bar{n}_{k+1}}{N_k}} \uparrow^j$. We prove by induction that $N_{k+1,j} \subseteq M_{k+1}$ for each j . $N_{k+1,0} = \emptyset$. So $N_{k+1,0} \subseteq M_{k+1}$. Assume $N_{k+1,j} \subseteq M_{k+1}$ for some $j \geq 0$. For any $a \in N_{k+1,j+1} - N_k$, $a \in T_{\frac{\bar{n}_{k+1}}{N_k}}(N_{k+1,j}) - N_k$. The predicate symbol occurring in a does not occur in $\bar{\Pi}_k$. (Otherwise, $a \in T_{\frac{\bar{n}_k}{N_k}} \uparrow \omega$, which is N_k by Lemma 6.7.) So there exists a clause $\alpha : a \leftarrow a_1, \dots, a_l$ in $\frac{\bar{n}_{k+1}}{N_k}$ such that each $a_i \in N_{k+1,j}$. So each $a_i \in M_{k+1}$. By Lemma 6.7, $\frac{\bar{n}_{k+1}}{N_k} = \frac{\bar{n}_{k+1}}{N_{k+1}}$. So α is also in $\frac{\bar{n}_{k+1}}{N_{k+1}}$. Let α be obtained from an instantiated clause $\beta : a \leftarrow a_1, \dots, a_l, \neg b_1, \dots, \neg b_m$ of Π_{k+1} . Then each $b_i \notin N_{k+1}$. We showed that $M_{k+1} \subseteq N_{k+1}$. So each $b_i \notin M_{k+1}$. Since each $a_i \in M_{k+1}$ and $\{a_1, \dots, a_l\}$ is finite, there exists a smallest number i' such that each $a_i \in M_{k+1,i'}$. Then $a \in T_{\Pi_{k+1}}(M_{k+1,i'})$. Therefore, $a \in M_{k+1}$. So $N_{k+1,j+1} \subseteq M_{k+1}$. Induction is completed. Since $N_{k+1} = T_{\frac{\bar{n}_{k+1}}{N_k}} \uparrow \omega = \bigcup_{j=0}^{\infty} N_{k+1,j}$, we have proved that $N_{k+1} \subseteq M_{k+1}$.

Thus $N_{k+1} = M_{k+1}$. The result is proved. $\square$

It has been shown by Lifschitz [39] that the standard model semantics is equivalent to a form of his *pointwise circumscription* [38]. The same result for *prioritized circumscription*, introduced by McCarthy [53] and further developed by Lifschitz [37], has been shown by Przymusiński [69]. Proposition 6.8 shows that the standard model semantics is equivalent to the application of the principle of setwise locality on the basis of the least Herbrand model semantics. Przymusiński proposed the *perfect model semantics* for *locally stratified program* [69]. Roughly speaking, a locally stratified program is a program such that its Herbrand instantiation can be stratified (probably infinitely). The principle of locality may also be applied to locally stratified programs, and the perfect model semantics may also be achieved on the basis of the least Herbrand model semantics.

6.3 Cumulative and Rational Monotony of Semantics

6.3.1 Kraus/Lehmann/Magidor's rules and Makinson's Operation

Kraus/Lehmann/Magidor studied a number of families of nonmonotonic consequence relations " $|\sim$ ". If α and β are formulas, then the expression $\alpha |\sim \beta$ (read *if α , normally β* , or *β is a plausible consequence of α*) has the meaning as following: if α is true, I am willing to (defeasibly) jump to the conclusion that β is true. They considered the following axiom and inference rules.

Reflexivity:

$$\alpha |\sim \alpha$$

Left logical Equivalence:

$$\frac{\models_{\mathcal{L}} \alpha \leftrightarrow \beta, \quad \alpha |\sim \gamma}{\beta |\sim \gamma}$$

Right Weakening:

$$\frac{\models_{\mathcal{L}} \alpha \rightarrow \beta, \quad \gamma |\sim \alpha}{\gamma |\sim \beta}$$

Cut:

$$\frac{\alpha \wedge \beta |\sim \gamma, \quad \alpha |\sim \beta}{\alpha |\sim \gamma}$$

Cautious Monotonicity:

$$\frac{\alpha \mid \sim \gamma, \quad \alpha \mid \sim \beta}{\alpha \wedge \beta \mid \sim \gamma}$$

Or:

$$\frac{\alpha \mid \sim \gamma, \quad \beta \mid \sim \gamma}{\alpha \vee \beta \mid \sim \gamma}$$

Rationality:

$$\frac{\alpha \not\mid \sim \neg \beta, \quad \alpha \mid \sim \gamma}{\alpha \wedge \beta \mid \sim \gamma}$$

The weakest logical system Kraus/Lehmann/Magidor defined is called **C** for *cumulative*. The system **C** consists of *Reflexivity*, *Left logical Equivalence*, *Right Weakening*, *Cut* and *Cautious Monotonicity*, and the corresponding consequence relation $\mid \sim$ is said to be *cumulative*. Left logical Equivalence and Right Weakening express the influence of the underlying logic $\mathcal{L}$. (In Kraus, Lehmann and Magidor's paper, $\models$ was used. However, the symbol $\models$ was defined as a relation between partial interpretations and literals in Definition 2.37 and 2.38.) For a formula α , $\models_{\mathcal{L}} \alpha$ means that α is a valid sentence in the underlying logic. Reflexivity and Right Weakening imply that $\alpha \mid \sim \beta$ if $\alpha \models_{\mathcal{L}} \beta$. They also defined the system **P**, which consists of **C** and *Or*, and the system **R**, which consists of **P** and *Rationality*. They gave various representation theorems for these systems in terms of their preferential model semantics [32].

The consequence relation $\mid \sim$ is defined between single formulae, and thus is finitistic. Makinson used an inference operation $C : 2^S \mapsto 2^S$ to study the cumulative inference in an infinitistic setting, where S is the set of all formulas of the language in question [44]. A *cumulative inference operation* is defined to be any operation satisfying the following conditions:

Reflexivity:

$$\alpha \subseteq C(\alpha)$$

Cumulativity:

$$\alpha \subseteq \beta \subseteq C(\alpha) \text{ implies } C(\alpha) = C(\beta)$$

In the above conditions, α and β are sets of formulae. A cumulative inference operator is closely related with the system **C** [32]. Intuitively, cumulativity says that adding a set of formulae inferable from a set of premises to the set of premises does not change the set of all formulae inferable from the premises. When considering infinite sets of formulae (for example the Herbrand instantiation of a program), Makinson's

terminology and approach seem more appropriate than " $|\sim$ ". The two notions C and " $|\sim$ " are connected via

$$\beta \subseteq C(\alpha) \text{ if and only if } \alpha |\sim \beta.$$

6.3.2 Cumulativity and Rationality

Not only can a logic program be taken as a knowledge base used to compute answers to queries against the base, but it can also be taken as an inference engine used to infer new conclusions in light of new information. Given a program, what conclusions are we willing to reach if α is true? The answer to this question depends not only on what α expresses, but also on what semantics we adopt for logic programming. Thus, for a program Π , each semantics SEM induces a relation $|\sim$ between formulas if we assume that new information is expressed by formulas. Then $\alpha |\sim \beta$ is intended to mean that if α is given, we infer β from the program Π under the semantics SEM . Furthermore, each semantics can also induce an inference operator C . For a set α of formulas, $C(\alpha)$ denotes the set of all formulas we are willing to conclude from the program under the semantics on the face of α .

The semantics of a logic program is given by one or a set of partial interpretations of the language. For a partial interpretation $I = \langle T, F \rangle$, T contains all the ground atoms that are interpreted as true atoms while F contains all the ground atoms that are interpreted as false atoms. Clearly, a partial interpretation may also be used to represent new information. Although a partial interpretation does not seem as expressive as an arbitrary formula or an arbitrary set of formulas, it is expressive enough to let us explain our ideas and give a uniform presentation.

In order to infer β from a program Π in light of new information α , we have to find a way to incorporate the new information with the inference engine represented by a program. We introduce the following notation.

Definition 6.8 Let $I = \langle T, F \rangle$ be a partial interpretation, and Π a normal program of a language $\mathbf{L}$. Then Π^F denotes the set obtained from $[\Pi]_{\mathbf{L}}$ by deleting each *nota* with $a \in F$ and $\Pi \oplus I$ is defined as $\Pi^F \cup T$.

Clearly, $\oplus$ is defined only for normal programs. For extended programs, $\oplus$ may be defined as $\Pi \oplus I = \Pi \cup T \cup \{\neg b : b \in F\}$. Study of $\oplus$ for normal programs is more interesting, and from now on, we only consider normal programs. Recall the incorporating operator $\uplus$ introduced in Definition 3.4. Let H be a hypothesis tuple and I a partial interpretation. Both $\Pi \oplus I$ and $H \uplus \Pi$ incorporate the negative information embedded in H and I the same way. However, $\uplus$ and $\oplus$ handle positive information differently. The difference is not essential when H is positively founded.

Given a program Π in a language $\mathbf{L}$, we define an operation on partial interpretations of $\mathbf{L}$ for any semantics SEM we adopt for logic programming.

$$C_{SEM}^{\Pi}(I) = \langle \{a : SEM(\Pi \oplus I) \models a\}, \{b : SEM(\Pi \oplus I) \models \neg b\} \rangle.$$

Clearly, $C_{SEM}^{\Pi}(I) = \cap_{M \in SEM(\Pi \oplus I)} M$. By Proposition 2.7, the intersection specifies the same semantics as SEM . Considering the connection between a cumulative inference operation C and a cumulative consequence relation " $\sim$ ", for any program Π , each semantics SEM also induces a " $\sim_{SEM}^{\Pi}$ " — relation between two partial interpretations, which is defined as follows.

$$I_1 \sim_{SEM}^{\Pi} I_2 \text{ iff } (I_2 \models l \text{ implies } SEM(\Pi \oplus I_1) \models l).$$

where l is a ground literal. With C_{SEM}^{Π} and " $\sim_{SEM}^{\Pi}$ " being so defined, they are connected via the same relation of C and " $\sim$ ", i.e.,

$$I_2 \subseteq C_{SEM}^{\Pi}(I_1) \text{ if and only if } I_1 \sim_{SEM}^{\Pi} I_2.$$

If $\models_C I_1 \leftrightarrow I_2$ and $\models_C I_1 \rightarrow I_2$ are interpreted as $I_1 = I_2$ and $I_2 \subseteq I_1$ respectively, $\sim_{SEM}^{\Pi}$ satisfies *Left Logical Equivalence* and *Right Weakening*. It should be noted that $\sim_{SEM}^{\Pi}$ and C_{SEM}^{Π} may not satisfy *Reflexivity*. For a partial interpretation I , $\Pi \oplus I$ is intended to mean the incorporation of Π and I , and the semantics of $\Pi \oplus I$ is to imply the information expressed by I . Therefore, $I \not\sim_{SEM}^{\Pi} I$ or $I \not\subseteq C_{SEM}^{\Pi}(I)$ amounts to the situation that the information expressed by I is not compatible with the information expressed by the program Π under the semantics SEM .

We formulate the infinitistic versions of *Cumulativity* and *Rationality*.

Cumulative Monotony:

$$\text{If } I_1 \subseteq I_2 \subseteq C_{SEM}^{\Pi}(I_1), \text{ then } C_{SEM}^{\Pi}(I_1) = C_{SEM}^{\Pi}(I_2).$$

Consider *Cautious Monotony* and *Cut*. The intended meaning of $\alpha \wedge \beta$ is that both α and β are true. Let I_1 and I_2 correspond to α and β respectively. Then $I_1 \wedge I_2$ can be interpreted as $I_1 \cup I_2$. With such an interpretation of $\wedge$, we get *Cautious Monotony* when " $=$ " is replaced by " $\subseteq$ "; and we get *Cut* when " $=$ " is replaced by " $\supseteq$ ". *Cautious Monotony* may be strengthened to *Rational Monotony*. Let $\neg I = \langle F, T \rangle$ if $I = \langle T, F \rangle$.

Rational Monotony:

$$\text{If } I_1 \subseteq I_2 \text{ and } \neg I_2 \cap C_{SEM}^{\Pi}(I_1) = \langle \emptyset, \emptyset \rangle, \text{ then } C_{SEM}^{\Pi}(I_1) \subseteq C_{SEM}^{\Pi}(I_2).$$

Rational Monotony is a stronger form of *Cautious Monotony* for any semantics because $\alpha \mid \sim_{SEM}^{\Pi} \beta$ implies $\alpha \mid \not\sim_{SEM}^{\Pi} \neg\beta$.

In [8, 10], Dix formulated *Cumulative Monotony* and *Rational Monotony* in a similar way. However, he considered only sets of positive atoms, i.e. in his formulation, I_1 and I_2 are replaced with sets U_1 and U_2 of atoms, which may be taken as partial interpretations $\langle U_1, \emptyset \rangle$ and $\langle U_2, \emptyset \rangle$. This is a special case of our framework. In his extension to his work, he considered sets of literals. His approach amounts to defining $\Pi \oplus I$ as $\Pi' \cup T$, where $I = \langle T, F \rangle$ and Π' is the program obtained from the Herbrand instantiation of Π by deleting every program clause with a head a which is in F . Such a definition of the incorporating operator $\oplus$ takes negative information embedded in I as facts and revise the program Π according to it. However, in the context of logic programming, negative information is inferred under certain assumption, or it is taken as hypotheses. Therefore, it should not be taken as facts and be used to revise a program. We believe that our definition of $\oplus$ is more appropriate for logic programming.

To illustrate the points of *Cumulative Monotony* and *Rational Monotony*, let us consider an example in a more general context than logic programming.

Example 6.3 [44, 8] Let *CIRC* be the circumscription [52, 53]. Then *CIRC* is cumulative but not rational. Let T be a first order theory and ϕ be a formulae true in every minimal model of T , i.e. $CIRC(T) \models_{c\ell} \phi$. Clearly, the minimal models of T are exactly the minimal model of $T \cup \{\phi\}$. Thus $CIRC(T) = CIRC(T \wedge \phi)$. In the case, the operator $\oplus$ is defined as the logical conjunction $\wedge$. Let T be the following program

$$\begin{aligned} \Pi : \quad & p \leftarrow \text{not} b \\ & b \leftarrow c \\ & c \leftarrow p, \text{not} a \end{aligned}$$

We have $CIRC(\Pi) \models_{c\ell} \text{not} c$ and " $CIRC(\Pi) \not\models \text{not} p$ ". However, $CIRC(\Pi \wedge p) \not\models \text{not} c$. Thus *Rational Monotony* is not satisfied by *CIRC*.

Definition 6.9 A semantics *SEM* is said to be *cumulative monotone* and *rational monotone* if the corresponding operation C_{SEM}^{Π} is cumulative and rational for any program Π .

Consider a semantics *SEM* which is cumulative monotone. Then for any program Π and any partial interpretation I , if $I \subseteq SEM(\Pi)$, then $SEM(\Pi) = SEM(\Pi \oplus I)$. Therefore, for a cumulative semantics, any information inferred from the knowledge base, represented by a program, can be incorporated with the knowledge base without changing the meaning of the knowledge base. From the point of view of implementation, cumulativity is the very property to guarantee an incremental implementation.

Unfortunately, not all the semantics we have discussed are cumulative monotone. In the rest of this section, we present some results on the cumulativity and rationality of various semantics.

In the following proposition and lemmas, L_m denotes the Lukasiewiczian m -valued logic with $m \geq 2$, Π is a normal program and $I = \langle T, F \rangle$, $I_1 = \langle T_1, F_1 \rangle$ and $I_2 = \langle T_2, F_2 \rangle$ are partial interpretations of a language L , $\Pi_1 = \Pi \oplus I_1 = \Pi^{F_1} \cup T_1$ and $\Pi_2 = \Pi \oplus I_2 = \Pi^{F_2} \cup T_2$. Lemma 6.10 and 6.11 are proved after Proposition 6.9.

Proposition 6.9

- (a) the L_m -well-founded semantics is cumulative and rational monotone;
- (b) the $\mathcal{L}$ -preferable semantics satisfies Cut, but is not cautious monotone, where $\mathcal{L}$ is either L_m or $\mathcal{AE}$.

Proof: Suppose $I_1 \subseteq I_2$. Note that $\Pi_1 = \Pi \oplus I_1$ and $\Pi_2 = \Pi \oplus I_2$.

Proof of 1. Let SEM denote SEM_{L_m-wf} . Let M_1 and M_2 be the L_m -well-founded models of Π_1 and Π_2 respectively. Then $C_{SEM}^\Pi(I_1) = M_1$ and $C_{SEM}^\Pi(I_2) = M_2$. We show that C_{SEM}^Π satisfies Cut. Suppose $I_2 \subseteq C_{SEM}^\Pi(I_1)$. If I is an L_m -saturated model of Π_1 , then $I_2 \subseteq I$ because M_1 is the least L_m -saturated model of Π_1 . By Lemma 6.10, I an L_m -saturated model of Π_2 . Therefore, $C_{SEM}^\Pi(I_1) \supseteq C_{SEM}^\Pi(I_2)$ because M_2 is the intersection of all L_m -saturated models of Π_2 . We then show SEM_{L_m-wf} is rational monotone. Suppose $\neg I_2 \cap C_{SEM}^\Pi(I_1) = \langle \emptyset, \emptyset \rangle$. Recall Definition 3.29 and the definition of L_m -well-founded model as the least L_m -saturated model. Let

$$M_{i0} = M_{i\perp};$$

$$M_{i\alpha+1} = \Psi_{\Pi_i}(M_{i\alpha});$$

$$M_{i\alpha} = \bigcup_{\beta < \alpha} \Psi_{\Pi_i}(M_{i\beta}), \text{ if } \alpha \text{ is a limit ordinal;}$$

where i is either 1 or 2, $M_{i\perp}$ is the least L_m -admissible model of Π_i and $M_{i\alpha} = \langle T_{i\alpha}, F_{i\alpha} \rangle$. We show by transfinite induction that $M_{1\alpha} \subseteq M_{2\alpha}$ for every ordinal α . Suppose $M_{1\beta} \subseteq M_{2\beta}$ for every $\beta < \alpha$. If α is a limit ordinal, clearly $M_{1\alpha} \subseteq M_{2\alpha}$. If α is not a limit, $\alpha - 1$ exists. For any $a \in F_{1\alpha}$, a is L_m -false-assumable of Π_1 wrt $M_{1\alpha-1}$. We show that a is L_m -false-assumable of Π_2 wrt $M_{2\alpha-1}$. For any $H_a = \langle P_a, N_a \rangle \subseteq \Delta_\Pi$ such that $H_a \uplus \Pi_2 \vdash_{L_m} a$, $H_a \uplus \Pi_1^{F_2} \cup T_2 \vdash_{L_m} a$ because $I_1 \subseteq I_2$. By Lemma 3.33, there exist $\{a_1, \dots, a_n\} \subseteq T_2$ and $\{b_1, \dots, b_k\} \subseteq B_a \cup F_2$ such that $\Pi_1 \vdash_{L_m} a \leftarrow a_1, \dots, a_n, \neg b_1, \dots, \neg b_k$. By Proposition 3.34, either (1) there exists a_i such that a_i is L_m -false-assumable of Π_1 wrt $M_{1\alpha-1}$, or (2) there exists b_j such that $\Pi_1^{F_{1\alpha-1}} \vdash_{L_m} b_j$. In case (1), $a_i \in F_{1\alpha}$. This contradicts $\neg I_2 \cap M_1 = \langle \emptyset, \emptyset \rangle$. In case (2), $b_j \in T_{1\alpha}$. Since $b_j \in F_2$ contradicts $\neg I_2 \cap M_1 = \langle \emptyset, \emptyset \rangle$, $b_j \in N_a$.

Because $I_1 \subseteq I_2$ and $M_{1\alpha-1} \subseteq M_{2\alpha-1}$ by assumption, $M_{2\alpha-1} \uplus \Pi_2 \vdash_{L_m} b_j$. Therefore, a is L_m -false-assumable of Π_2 wrt $M_{2\alpha-1}$. Hence $a \in F_{2\alpha}$ and $F_{1\alpha} \subseteq F_{2\alpha}$. Since $I_1 \subseteq I_2$, $M_{1\alpha} \subseteq M_{2\alpha}$. We proved $M_{1\alpha} \subseteq M_{2\alpha}$ for every ordinal α . Since there exists δ_i such that $M_i = M_{i\gamma}$ for any $\gamma \geq \delta_i$, where i is either 1 or 2, $M_1 \subseteq M_2$. Thus $C_{SEM}^\Pi(I_1) \subseteq C_{SEM}^\Pi(I_2)$.

Proof of 2. Let SEM denote SEM_{L_m-p} . Suppose $I_2 \subseteq C_{SEM}^\Pi(I_1)$. If I is an L_m -preferable model of Π_1 , then $I_2 \subseteq I$. By Proposition 3.11, I is also L_m -saturated of Π_1 . By Lemma 6.10, I is L_m -saturated of Π_2 . We then show that I is L_m -preferable model of Π_2 . Let $I' = \langle T', F' \rangle$ be an L_m -preferable model of Π_2 and $I \subseteq I'$. Because $I_2 \subseteq I$ and I is L_m -saturated of Π_1 , $\Pi_1^F \vdash_{L_m} a$ for each $a \in T_2$. Thus $\Pi_1^{F'} \vdash_{L_m} a$ for each $a \in T_2$ since $F \subseteq F'$. By Lemma 6.11, I' is L_m -saturated of Π_1 . Since I is L_m -preferable of Π_1 and $I \subseteq I'$, $I = I'$. Therefore, I is L_m -preferable of Π_2 . Thus $C_{SEM}^\Pi(I_1) \supseteq C_{SEM}^\Pi(I_2)$.

Let SEM denote $SEM_{\mathcal{AE}-p}$. Suppose $I_2 \subseteq C_{SEM}^\Pi(I_1)$. If I is an $\mathcal{AE}$ -preferable model of Π_1 , then $I_2 \subseteq I$. Let i be either 1 or 2.

- I is $\mathcal{AE}$ -preferable of Π_i
- iff I is a stable model of Π_i by Theorem 4.16
- iff I is an L_3 -stable model of Π_i by Theorem 4.14
- iff I is an L_3 -preferable model of Π_i and I is total.

By paraphrasing the proof in the last paragraph, we can prove that every total L_m -preferable model of Π_1 is L_m -preferable of Π_2 . Therefore, every $\mathcal{AE}$ -preferable model of Π_1 is $\mathcal{AE}$ -preferable of Π_2 . Thus, $C_{SEM}^\Pi(I_1) \supseteq C_{SEM}^\Pi(I_2)$. We just proved that the $\mathcal{L}$ -preferable semantics satisfies Cut. The next example shows it is not cautious monotone. $\square$

Lemma 6.10 *Let I be an L_m -saturated model of Π_1 . If $I_1 \subseteq I_2 \subseteq I$, then I is an L_m -saturated model of Π_2 .*

Proof: The result follows from Proposition 3.35 by verifying that I satisfies the condition (a), (b) and (c) for Π_2 . $\square$

Lemma 6.11 *Let I be an L_m -saturated model of Π_2 . If $I_1 \subseteq I_2 \subseteq I$ and $\Pi_1^F \vdash_{L_m} a$ for each $a \in T_2$, then I is an L_m -saturated model of Π_1 .*

Proof: T and F are disjoint by Proposition 3.35 because I is L_m -saturated of Π_1 . For any atom a ,

$a \in T$

iff there exists $\gamma : a \leftarrow \text{not}b_1, \dots, \text{not}b_n$ such that $\Pi_2 \vdash_{L_m} \gamma$, where $b_i \in F$, by Proposition 3.35

iff $\Pi_2^F \vdash_{L_m} a$ by Lemma 3.33

iff $\Pi_1^F \cup T_2 \vdash_{L_m} a$

iff $\Pi_1^F \vdash_{L_m} a$ because $\Pi_1^F \vdash_{L_m} b$ for each $b \in T_2$

iff there exists $\gamma' : a \leftarrow \text{not}b'_1, \dots, \text{not}b'_n$ such that $\Pi_1 \vdash_{L_m} \gamma'$, where $b'_i \in F$ by Lemma 3.33.

I satisfies the condition (b) of Proposition 3.35.

$a \in F$

iff for any $\gamma : a \leftarrow a_1, \dots, a_m, \text{not}b_1, \dots, \text{not}b_n$ such that $\Pi_2 \vdash_{L_m} \gamma$, there exist $a_i \in F$ or $b_j \in T$ by Proposition 3.35

iff for any $\gamma' : a \leftarrow a'_1, \dots, a'_m, \text{not}b'_1, \dots, \text{not}b'_n$ such that $\Pi_1 \vdash_{L_m} \gamma'$, there exist $a'_i \in F$ or $b'_j \in T$ by Lemma 3.33 because T and F are disjoint and $I_2 \subseteq I$.

Therefore, I is L_m -saturated of Π_1 by Proposition 3.35. $\square$

Example 6.4 Consider the following program Π taken from [92, Example 5.1].

$$\begin{aligned} a &\leftarrow \text{not}b \\ b &\leftarrow \text{not}a, c \\ c &\leftarrow a \end{aligned}$$

Let $I_1 = \langle \emptyset, \emptyset \rangle$ and $I_2 = \langle \{c\}, \emptyset \rangle$. Since $I = \langle \{c, a\}, \{b\} \rangle$ is the only $\mathcal{L}$ -preferable model of Π , $I_2 \subseteq SEM_{\mathcal{L}-p}(\Pi \oplus I_1) = SEM_{\mathcal{L}-p}(\Pi) = I$. However, $\langle \{c, b\}, \{a\} \rangle$ is also $\mathcal{L}$ -preferable of $\Pi \oplus I_2$. Thus $C_{SEM_{\mathcal{L}-p}}^\Pi(I_1) \not\subseteq C_{SEM_{\mathcal{L}-p}}^\Pi(I_2)$. Therefore, $SEM_{\mathcal{L}-p}$ is not cautious monotone.

Why is $\langle \{c, b\}, \{a\} \rangle$ not $\mathcal{L}$ -preferable of Π ? The very reason to infer c from the program is that $\neg b$ is accepted as a hypothesis and it is the only acceptable hypothesis according to our criterion, admissibility. That is to say that c is inferable because the hypothesis $\neg b$ is admissible. If we add c to the program, such a dependence of c on $\neg b$ is lost, and now c is inferable from the program independently. Furthermore, adding c to the program makes $\neg a$ admissible. Therefore, $\neg b$ serves as a justification for c to be inferred from the program. It is far from obvious that these cases actually

should yield the same results if the justifying information for inferring something, i.e. the hypothesis and the dependence are taken into account. It will turn out that those semantics have the property of cumulativity and even rational monotony if such justifications are required to be embedded in the partial interpretation to be incorporated with a program. For instance, if $I_2 = \langle \{c\}, \{b\} \rangle$, then Π and $\Pi \oplus I_2$ gave exactly the same $\mathcal{L}$ -preferable model. This observation also suggests that the reason for the cumulativity of the L_m -well-founded semantics is that the semantics can be defined as an iterated fixed point starting from the least L_m -admissible model $M_\perp$. Thus everything inferable under the semantics can be justified by itself. Therefore, incorporating information inferred from the program under the semantics would not change the meaning of the program.

The set of all $\mathcal{L}$ -admissible models of a normal program is a class of partial interpretations which have the justifying information embedded in themselves.

Proposition 6.12 *Let Π be a normal program and I an L_m -admissible model of Π and SEM denote any of the L_m -well-founded, the L_m -preferable or the $\mathcal{AE}$ -preferable semantics. Then $I \subseteq C_{SEM}^\Pi(I)$.*

Proof: Note that $\mathcal{AE}$ -preferable hypothesis tuple is a L_3 -stable tuple. Thus we need only consider L_m in the following proof. Let $I = \langle T, F \rangle$ and $I' = \langle T', F' \rangle$ be an L_m -saturated model of $\Pi \oplus I$. It can be shown that each $a \in F$ is L_m -false-assumable of $\Pi \oplus I$ wrt I' . Then $I \subseteq I'$ because $\Pi^F \vdash_{L_m} a$ implies $(\Pi \oplus I)^{F'} \vdash_{L_m} a$ for any atom a . It follows immediately that $I \subseteq C_{SEM}^\Pi(I)$ since every partial interpretation in $SEM(\Pi \oplus I)$ is an L_m -saturated model of $\Pi \oplus I$. $\square$

Therefore, C_{SEM}^Π satisfies Reflexivity when it is restricted to admissible models. Actually, it is reflexive when restricted to any partial interpretation $I = \langle T, \emptyset \rangle$. This observation suggests that Reflexivity itself is not able to justify the partial interpretation as indicated by Example 6.4. Roughly speaking, reflexivity may provide justification for the negative information expressed by the admissible model, not necessarily for positive information. Another property an $\mathcal{L}$ -admissible model $I = \langle T, F \rangle$ of a program Π has is that for each $a \in T$, $\Pi^F \models_{\mathcal{L}} a$. This property provides justification for positive information expressed by the admissible model. These two properties together guarantee that the justifying information is embedded in the partial interpretation.

Definition 6.10 Let Π be a program and SEM be a semantics. A partial interpretation $I = \langle T, F \rangle$ is said to be *justified* wrt $SEM(\Pi)$ in logic $\mathcal{L}$ if I satisfies the following two conditions:

- (a) $I \subseteq C_{SEM}^\Pi(I)$; and

(b) $\Pi^F \models_{\mathcal{L}} a$ for each $a \in T$.

Proposition 6.13 *Let SEM be $\mathcal{L}$ -preferable semantics, where $\mathcal{L}$ is either L_m or $\mathcal{AE}$, and Π be a program. If C_{SEM}^{Π} is restricted to partial interpretation justified wrt $SEM(\Pi)$ in logic $\mathcal{L}$, it satisfies Rational Monotony.*

Proof: Suppose $I_1 \subseteq I_2$. Note that $\Pi_1 = \Pi \oplus I_1$ and $\Pi_2 = \Pi \oplus I_2$. Let I be an $\mathcal{L}$ -preferable model of Π_2 . Then I is L_m -saturated of Π_2 . (When $\mathcal{L}$ is $\mathcal{AE}$, I is L_3 -stable, and hence L_m -preferable and L_m -saturated of Π_2 .) Because I_2 is justified, $I_2 \subseteq I$ and $\Pi^{F_2} \models_{L_m} a$ for each $a \in T_2$. Then $\Pi^F \models_{L_m} a$ for each $a \in T_2$. Since $\Pi_1 \models_{L_m} \Pi$, $\Pi_1^F \models_{L_m} a$ for each $a \in T_2$. By Lemma 6.11, I is L_m -saturated of Π_1 . Since every $\mathcal{L}$ -saturated model of Π_1 is L_m -saturated of Π_2 by Lemma 6.10 and I is L_m -preferable of Π_2 , I is L_m -preferable of Π_1 . (When $\mathcal{L}$ is $\mathcal{AE}$, I is an L_3 -stable model of Π_1 and hence is an $\mathcal{AE}$ -preferable model of Π_1 .) Thus $C_{SEM}^{\Pi}(I_1) \subseteq C_{SEM}^{\Pi}(I_2)$. $\square$

Proposition 6.9 shows that the $\mathcal{L}$ -preferable semantics satisfies Cut. Thus by the above proposition, $\mathcal{L}$ -preferable semantics is cumulative when restricted to justified partial interpretations. This result is not a surprise. It has been shown that the default logic defined by Reiter [76] is not cumulative [44, 4]. By keeping track of justifications for deriving something from a default theory, Brewka defined cumulative default logic CDL [4]. Instead of simple first-order formulae this logic uses more complex structure, called *assertions*, to represent theorems of a default theory. An assertion is of the form $\langle p : \{r_1, \dots, r_n\} \rangle$. The intuitive meaning of the assertion is: p is believed under the justification $\{r_1, \dots, r_n\}$. I.e. p is believed, since $r_1 \wedge \dots \wedge r_n$ is consistent with what is believed and the justifications of other believed formulae. The notion of justified partial interpretations follows the idea of the notion of assertions.

6.4 Supraclassicality

Besides reflexivity and cumulativity for a cumulative inference operator C , Makinson also defined two other desirable properties for C .

Supraclassicality:

$$Cn(\alpha) \subseteq C(\alpha)$$

Distributivity:

$$C(\alpha \cup \beta_1) \cap C(\alpha \cup \beta_2) \subseteq C(\alpha \cup (Cn(\beta_1) \cap Cn(\beta_2)))$$

In the above, Cn is the operation of tautological consequence as applied to the propositions in S , noting that S may or may not involve a functionally complete set of classical connectives, and may perhaps use certain connectives without being closed under them [44]. With the connection of C and " $\sim$ " being that $\beta \subseteq C(\alpha)$ iff $\alpha \sim \beta$, a cumulative inference operation C satisfying *Reflexivity*, *Cumulativity*, *Supraclassicality* and *Distributivity* corresponds to the consequence relation $\sim$ of system **P** [8].

In the next two sections, we will consider the two properties *Supraclassicality* and *Distributivity*, and formulate the two properties for the operator Cn and C_{SEM}^Π in the context of logic programming. Let us recall the definition of supraclassicality: $Cn(\alpha) \subseteq C(\alpha)$. The operation Cn of tautological consequence is applied to the propositions in the set S of all formulae. In our framework for semantics of logic programming, the operation C_{SEM}^Π for any program Π and semantics SEM is a mapping among partial interpretations, and all the semantics defined in our framework are based on the semantics of an underlying logic $\mathcal{L}$. We define the operation Cn as the logical consequence relation of $\mathcal{L}$. Let Π be a program and I be a partial interpretation.

$$Cn_{\mathcal{L}}^\Pi(I) = \langle \{a : \Pi \oplus I \models_{\mathcal{L}} a\}, \{b : \Pi \oplus I \models_{\mathcal{L}} \neg b\} \rangle,$$

where a and b are ground atoms. When $\mathcal{L}$ is the autoepistemic logic $\mathcal{AE}$, Π is considered a theory in $\mathcal{AE}$ logic. Accordingly, the definition of $Cn_{\mathcal{AE}}^\Pi$ is changed as follows:

$$Cn_{\mathcal{AE}}^\Pi(I) = \langle \{a : \Pi \oplus I(L) \models_{\mathcal{AE}} a\}, \{b : \Pi \oplus I(L) \models_{\mathcal{AE}} \neg b\} \rangle.$$

When $\mathcal{L}$ is the Lukasiewiczian m -valued logic L_m , $\Pi \oplus I \not\models_{\mathcal{L}} \neg b$ for any b . Thus $F = \emptyset$ if $Cn_{L_m}^\Pi(I) = \langle T, F \rangle$ for any I . With $Cn_{\mathcal{L}}^\Pi$ so defined, we formulate the Supraclassicality wrt $\mathcal{L}$ for a semantics SEM , i.e. for the operator C_{SEM}^Π .

Supraclassicality wrt $\mathcal{L}$:

$$Cn_{\mathcal{L}}^\Pi(I) \subseteq C_{SEM}^\Pi(I)$$

Definition 6.11 A semantics SEM is said to be *supraclassical wrt a logic $\mathcal{L}$* if the corresponding operators $Cn_{\mathcal{L}}^\Pi$ and C_{SEM}^Π satisfy supraclassicality wrt $\mathcal{L}$ for any program Π .

The rationale behind supraclassicality is the following. A logic program Π is considered a set of sentences of certain underlying logic $\mathcal{L}$. Then, any conclusion

that can be drawn from a program under the semantics of the logic $\mathcal{L}$ should also be inferable from the program under the semantics of logic programming. In other words, the semantics of logic programming should be able to capture the semantics of the underlying logic. A similar property was called closedness under logical consequence in [28].

Proposition 6.14 *The L_m -well-founded and the L_m -preferable semantics are supra-classical wrt L_m . The $\mathcal{AE}$ -preferable semantics is supraclassical wrt $\mathcal{AE}$.*

Proof: Let Π be any program and I any partial interpretation. Let SEM denote SEM_{L_m-wf} or SEM_{L_m-p} . $Cn_{L_m}^\Pi(I) \subseteq C_{SEM}^\Pi(I)$ follows from the fact that $\Pi \oplus I \models_{L_m} a$ implies $(\Pi \oplus I)^{F'} \models a$ for any L_m -saturated model $\langle T', F' \rangle$ of $\Pi \oplus I$.

Let SEM denote $SEM_{\mathcal{AE}-p}$. By Theorem 4.16, $\Pi \oplus I(L) \models_{\mathcal{AE}} l$ iff $SEM(\Pi \oplus I) \models l$. Thus $Cn_{\mathcal{AE}}^\Pi(I) = C_{SEM}^\Pi(I)$. $\square$

6.5 Principle of Proof by Cases

Let us go back to the discussion about the properties of the cumulative consequence relation $|\sim$. Makinson suggested the following rule, which expresses the principle of proof by cases.

Proof by Cases:

$$\frac{\alpha \wedge \neg\beta \mid\sim \gamma, \quad \alpha \wedge \beta \mid\sim \gamma}{\alpha \mid\sim \gamma}$$

It can be shown that the *Proof-by-Cases* rule is equivalent to the rule *Or* in the presence of the system C [32]. When the cumulative inference operation C and the cumulative consequence relation " $|\sim$ " are connected via $\beta \subseteq C(\alpha)$ iff $\alpha \mid\sim \beta$, *Distributivity* corresponds to the rules *Or* and *Left Logical Equivalence* [8]. In this section, we formulate the principle of proof by cases for the semantics of logic programming in our framework.

Example 6.5 Let us consider the following program Π .

```
a ← not b
b ← not a
d ← a
d ← not a
```

The *Proof-by-Cases* rule says that if a formula γ is provable (or derivable) in both cases when another formula β is true and when it is false, then the formula γ is

provable without β being a premise. In the above program, $d \leftarrow a$ means that d is provable when a is proved, and $d \leftarrow \text{nota}$ means that d is provable when a is not provable, or is justifiably assumed false. According to the *Proof-by-cases* rule, d should be provable from the program, and it does not matter whether a is provable from the program or not.

In general, we should consider all cases generated by a set S of ground atoms, instead of only the two cases a and $\neg a$ of a single atom. An atom in S can be either true or false. Thus a case of S can be represented by a partial interpretation $I_c = \langle c, S - c \rangle$, where $c \in 2^S$. In a case I_c , c contains all atoms that are true and $S - c$ contains all the atoms that are false when only atoms in S are concerned. For a set S of atoms, we write S_{case} to denote all the partial interpretation representing all the cases of S .

The symbol $\leftarrow$ in a program clause $a \leftarrow l_1, \dots, l_n$ of a program Π expresses the provability (or derivability) of a from $l_1, \dots, l_n$. Such a notion of provability is characterized by $\vdash_{\text{Horn}}$ defined in Definition 3.36. Naturally, we have the following definition.

Definition 6.12 A semantics SEM is said to obey the *Principle of Proof-by-Cases* if for any program Π and any set S of ground atoms, $\Pi \oplus I_c \vdash_{\text{Horn}} a$ for every case $I_c \in S_{\text{case}}$ implies $SEM(\Pi) \models a$, where a is an ground atom.

Proposition 6.15

- (a) Neither L_3 -well-founded nor L_3 -preferable semantics obeys the Principle of Proof-by-Cases;
- (b) Both L_2 -well-founded and L_2 -preferable semantics obey the Principle of Proof-by-Cases;
- (c) $\mathcal{AE}$ -preferable semantics obeys the Principle of Proof-by-Cases.

Proof: (a) is proved by the following Example 6.6.

Let Π be a program and a a ground atom.

Proof of (b): We show that $\Pi \models_{L_2} a$ if there exists a set S of ground atoms such that $\Pi \oplus I_c \vdash_{\text{Horn}} a$ for every case $I_c \in S_{\text{case}}$. Suppose $\Pi \oplus I_c \vdash_{\text{Horn}} a$ for every $I_c \in S_{\text{case}}$. Then there exists $\gamma' : a \leftarrow l'_1, \dots, l'_n$ such that $\Pi \vdash_{\text{Horn}} \gamma'$ and $I_c \models l'_i$ by Lemma 3.33. Clearly, $\Pi \vdash_{\text{case}} \gamma'$. By Lemma 3.28, $\Pi \models_{L_2} \gamma'$. For any (2-valued) interpretation I , there exists a case I_c such that $I_c \models l$ implies $I/I = \mathbf{t}$ for any ground literal l . Therefore, there exists γ' such that $l'_1 \wedge \dots \wedge l'_n / I = \mathbf{t}$. If I is a model for Π , it is a model for γ' too. Thus $I/a/I = \mathbf{t}$. Therefore, $\Pi \models_{L_2} a$. Then the result follows from immediately.

Proof of (c): Let $I = \langle T, F \rangle$ be an $\mathcal{AE}$ -preferable model of Π . It is a stable model of Π by Theorem 4.16. And it is an L_3 -stable model of Π_i by Theorem 4.14. Thus it is an L_3 -preferable model of Π_i by Proposition 3.12. Suppose there exists a set S of ground atoms such that $\Pi \oplus I_c \vdash_{Horn} a$ for every case $I_c \in S_{case}$. There exists a case $I_c = \langle T_c, F_c \rangle \in S_{case}$ such that $I_c \subseteq I$. Then $\Pi^F \models_{L_3} b$ for every $b \in T_c$. Thus $\Pi^F \models_{L_3} \Pi \oplus I_c$ since $F_c \subseteq F$, and $\Pi^F \models_{L_3} a$. Therefore, $a \in T$ and $I \models a$. We proved $SEM_{\mathcal{AE}-p}(\Pi) \models a$. $\square$

Example 6.6 Consider the following program Π .

$$\begin{aligned} p &\leftarrow a \\ p &\leftarrow c \\ c &\leftarrow \neg a \\ b &\leftarrow \neg c \\ a &\leftarrow \neg b \end{aligned}$$

The L_3 -well-founded model of Π is $\langle \emptyset, \emptyset \rangle$, which is also the unique L_3 -preferable model of Π . Let $S = \{a\}$. Then $\Pi \oplus I_c \vdash_{Horn} p$ for the two cases of $\{a\}$, but $SEM_{L_3-wf}(\Pi) \not\models p$ and $SEM_{L_3-p}(\Pi) \not\models p$.

Chapter 7

Related Work and Future Research

7.1 Related Work

Framework: Przymusiński introduced *Autoepistemic Logics of Closed Beliefs* AEL_{NMF} , where NMF denotes a specific *nonmonotonic formalism* on which the logic is based [74]. He showed that Moore's autoepistemic logic $\mathcal{AE}$ is a *special case* of AEL_{NMF} obtained when Reiter's Closed World Assumption (CWA) is used as NMF . Since the stable model semantics is based on Moore's logic $\mathcal{AE}$, it can be defined by AEL_{CWA} . When NMF is *Parallel Circumscription* $CIRC$, he showed that for any program Π the semantics given by the least stable autoepistemic expansion of Π in AEL_{CIRC} coincides with the well-founded semantics [74, Theorem 3.3]. So his work provides a framework for the stable model and well-founded model semantics in logic programming. While his work is based on various formalisms for nonmonotonic reasoning, such as $\mathcal{AE}$, CWA and $CIRC$, our framework is based only on abduction and is simpler and more concise. Another advantage of our work is that our framework is applicable to more semantics than AEL_{NMF} .

Using a model theoretical approach, Dix proposed a semantical framework in terms of 3-valued models that have the properties P1-P4, presented in Section 3.3.2 [10]. The semantical models are defined as the minimal elements of the 3-valued models with respect to certain binary relations. The framework successfully represents Fitting's 3-valued predicate completion, which our framework could not represent, the well-founded model semantics and WFS' (which is a semantics defined in [10]). The other semantics WFS^+ defined in [10] has been proved to coincide with the L_2 -well-founded semantics [11].

Using alternating fixed point theory, You and Yuan showed the equivalence of

some well-known semantics for normal programs [95].

Principle of Locality: Besides the principle of locality, which is referred to as the principle of relevance, Dix also proposed other principles, including the *principle of partial evaluation (PPE)*, *modularity*, *reduction* and *isomorphy* [10]. PPE, modularity and reduction reflect different aspects of the principle that (1) the meaning of atoms should be locally determined by their relevant program; and (2) the semantics of the program obtained by incorporating the meaning of the atoms should be the same as the original program. Dix also extended his work to disjunctive programs [9].

Based on the principle of locality, Wasserman and Shen suggested an alternative approach to the completeness problem of Clark's predicate completion [88]. In general, Clark's semantics does not satisfy the principle of locality and its corresponding procedural semantics SLDNF-resolution is not complete. They provided more justifications for the principle of locality by showing that Clark's semantics satisfies the principle of locality exactly when the completeness of SLDNF-resolution, which always satisfies the principle, is achieved.

Abduction: Eshghi and Kowalski introduced the notion *abductable predicate* to establish the relationship of abduction and logic programming [15], and Dung used the same approach [12]. In their work, only abductable atoms, which are substituted in logic programs for negative literals, are considered possible hypotheses. Dung's work has been reviewed in Section 4.2. Li and You introduced another abduction-based semantics [35]. In their work, both abductable and ordinary atoms are considered possible hypotheses. The semantics is recursively defined with two concepts, the abductive and the hypothetical implication, which are used to deduce ordinary and abductable atoms respectively. Their semantic is different from Dung's preferential extension semantics and any of the semantics in our framework.

7.2 Suggestions for Future Research

The results of this dissertation contribute in establishing the basis for future research in the field of logic programming. In this section, we suggest several topics for future research.

7.2.1 Procedural Semantics

Besides a declarative semantics, logic programming also needs a proof theory which specifies how conclusions are to be derived from a logic program. Such a proof theory is also said to specify the procedural semantics of logic programming. Two indispensable notions about a procedural semantics are the soundness and the completeness

with respect to a certain declarative semantics. A procedural semantics is sound if it derives only conclusions that are inferable from the concerned declarative semantics; it is complete if it derives all conclusions that are inferable from the concerned declarative semantics. Another important notion about a procedural semantics is the effectiveness. A procedural semantics is effective if the corresponding proof theory is effective. (A proof theory is effective if there exists a mechanical proof process that terminates.)

Enormous work has been done to study the soundness and completeness issue of proof theories for various declarative semantics. SLD-resolution is sound and complete with respect to the least Herbrand model semantics [7, 42]. SLDNF-resolution is sound with respect to Clark's predicate completion [42], but complete only under certain conditions [5, 34, 88]. SLDNF-resolution was extended to SLS-resolution (SL-resolution for Stratified program) [67], and further extended to global SLS-resolution for the well-founded semantics [79]. Global SLS-resolution is sound and complete with respect to the well-founded semantics for non-floundering queries, but it is not effective because the termination of the procedure (sometimes) depends on the nontermination of the global SLS-derivation, by which the resolution is defined. Nevertheless, it is argued in [67, 79] that such a procedure may be considered a theoretical construct, an ideal query answering procedure to which various effective approximation may be compared.

In contrast, relatively little work has been done on the proof theory for semantics based on abduction. Dung gave a soundness result of the abduction procedure proposed by Eshghi and Kowalski [15] with respect to his semantics [12]. Li and You generalized Eshghi and Kowalski's algorithm, and showed the soundness of the generalization with respect to their abductive deduction semantics [35]. More work needs to be done to study the proof theoretic aspects of abduction in the context of logic programming.

Most of the proof theory proposed for various declarative semantics are not decidable, and thus the procedures or algorithms are not effective. The reason for this is that, in general, the relations or sets that one can characterize through the semantics are not recursively enumerable. Therefore, we should not expect to have an effective proof theory which is also sound and complete with respect to the corresponding declarative semantics. However, under certain conditions, an ineffective procedure may be rendered an effective one. One of such conditions is the *bounded term size* property defined by Van Gelder in [22] for the *top-down-positive derivation*. Informally speaking, a logic program has the bounded term size property if there is a function $f(n)$ such that whenever the original query Q has no argument whose term size exceeds n , then the term size of any argument involved in the processing of Q

does not exceed $f(n)$. (The size of a term is the count of its functors, constants, and variables.) Van Gelder defined an effective procedural semantics that coincides with the perfect model semantics when the program is stratified and has the property. Although it is semantically meaningless, the term size bounded property is an important condition for the effectiveness of proof theories of logic programming because it guarantees a finite search domain for a procedure. A deductive database in which there is no function symbol satisfies this property. It remains to be seen whether it is possible to find other conditions that are weaker but still significant enough to guarantee the effectiveness of a proof theory.

7.2.2 Equivalence of Logic Programs

The equivalence relationship is at the basis of most programming methodologies. Since there are different methods to define a semantics for a logic programs, different equivalences may possibly be induced by each different method. Without a clear understanding of both the declarative semantics and the procedural semantics of logic programming, it is hard, if not impossible, to establish the equivalence relation on logic programs, and to investigate how different equivalences are related. With both the declarative semantics and the procedural semantics of definite programs being well understood, Maher systematically investigated the ideas of equivalence of definite programs, which consist of Horn clauses [43]. Maher compared the relative strengths of various equivalence formulations, which arise naturally from several well-known semantics including both declarative and procedural semantics. (Maher used different terminology, such as the logical semantics, the operational semantics and the functional semantics.) To the best of our knowledge, there is no similar work for logic programs with negation so far. Results in this dissertation would provide a better understanding of the semantics for logic programming, and thus makes it possible to conduct the equivalence research in the context of logic programs allowing negation.

7.2.3 Update Semantics

As a tool for representing knowledge, a logic program is intended to be a knowledge base. We expect that the knowledge in a representation system should be easy to change. It gives rise to a very important issue for knowledge bases, the semantics of update. The update semantics for a knowledge base based on logic programming is not clear yet although some research has been done on updating base relations or predicates (which do not occur in the heads of any rules,) and on update languages [46, 62, 45]. There are two approaches to the issue, the syntactical approach and the model-theoretic approach. By the syntactical approach, the update semantics is

characterized by the minimal change of the theory representing the knowledge base [17]. Gardenfors, Alchourron and Makinson have done fundamental work on the issue [20, 1, 21]. They propose, on philosophical grounds, a set of rationality postulates for the update semantics from the proof-theoretical point of view. Winslett proposed a model-theoretic approach [89, 90, 91]. By the model-theoretical approach, the update semantics is characterized by the minimal change of the models, which corresponds to the possible worlds represented by the theory, rather than by the minimal change of the theory, which is merely the description of the possible worlds. Unfortunately, as indicated by Winslett, a minimal change in the world does not necessarily correspond to a minimal change in the description of the world, or vice versa. While one approach makes great sense from its own point of view, it does not make proper sense from the other approach's point of view. The relationship between these two approaches has not been established. Other important points worth noting are that in the research mentioned above, the logical force of the knowledge base is considered and the formation of the base is ignored. The syntactical form is ignored in the proof theoretical approach, and all models are taken into account in the model-theoretical approach. These features do not fit in the context of logic programming, where syntax does bear an intended meaning, and only some of the models are viewed as intended models, and thus should be taken into account. With well defined declarative semantics as a result of this dissertation, we provide a basis for developing an update semantics for knowledge bases in the context of logic programming.

Bibliography

- [1] C.E. Alchourron, P. Gardenfors, and D. Makinson. On the logic of theory change: partial meet contraction and revision. *Journal of Symbolic Logic*, 50:510-530, 1985.
- [2] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89-142. Morgan Kaufmann Publisher, Los Altos, CA, 1988.
- [3] K. Apt and M.H. Van Emden. Contributions to the theory of logic programming. *JACM*, 29(3):841-862, July 1982.
- [4] G. Brewka. Cumulative default logic: in defense of nonmonotonic inference rules. *Artificial Intelligence*, 50:183-205, 1991.
- [5] L. Cavedon and J. Lloyd. A completeness theorem for sldnf-resolution. *Journal of Logic Programming*, 7(3):177-192, 1989.
- [6] Jianhua Chen. Minimal knowledge + negation as failure = only knowing (sometimes). In *Proc. of 10th International Conference of Logic Programming*, 1993.
- [7] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293-322. Plenum Press, New York, 1978.
- [8] J. Dix. Classifying semantics of logic programs. In *Proc. of 1st Int'l. Workshop on Logic Programming and Non-monotonic Reasoning*, pages 166-180, 1991.
- [9] J. Dix. Classifying semantics of disjunctive logic programs. In *Logic Programming: Proc. of 1992 Joint Int'l. Conf. and Symposium*, 1992.
- [10] J. Dix. A framework for representing and characterizing semantics of logic programs. In *Principle of Knowledge and Representation and Reasoning: Proc. 3rd Intl. Conf.*, 1992.
- [11] J. Dix. A classification-theory of semantics of normal logic program: I strong properties. Technical Report TR 10/93, University of Koblenz, Dept. of Computer Science, September 1993.

- [12] P.M. Dung. Negation as hypotheses: an abductive foundation for logic programming. In *Proc. of 8th International Conference of Logic Programming*, pages 3-17, 1991.
- [13] P.M. Dung. An argumentation semantics for logic programming with explicit negation. In *Proc. of 10th International Conference of Logic Programming*, pages 616-630, 1993.
- [14] M.H. Van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *JACM*, 23(4):733-742, Oct. 1976.
- [15] K. Eshghi and R.A. Kowalski. Abduction compared with negation as failure. In *Proc. of 6th International Conference of Logic Programming*, pages 234-254, 1989.
- [16] C. Hartshorn et al, editor. *The Collected Papers of Charles Sanders Peirce*. Harvard University Press, 1931.
- [17] R. Fagin, J.D. Ullman, and M.Y. Vardi. On the semantics of updates in databases. In *ACM 2th Symposium on PODS*, pages 325-365, 1983.
- [18] M. Fitting. A kripke-kleene semantics for logic programs. *The Journal of Logic Programming*, 2(4):295-312, 1985.
- [19] D. Gabbay. Theoretical foundations for non-monotonic reasoning in expert systems. In K.R.. Apt, editor, *Logics and Models of Concurrent systems*, pages 439-457. Springer-Verlag, Berlin, 1985.
- [20] P. Gardenfors. Epistemic importance and minimal changes of belief. *Australasian Journal of Philosophy*, 62:136-157, 1984.
- [21] P. Gardenfors and D. Makinson. Revisions of knowledge systems using epistemic entrenchment. In *Proc. of the 2nd Conf. on Theoretical Aspects of Reasoning about Knowledge*, pages 83-95, 1988.
- [22] A. Van Gelder. Negation as failure using tight derivations for general logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 149-176. Morgan Kaufmann Publisher, Los Altos, CA, 1988.
- [23] A. Van Gelder, K. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *ACM 7th Symposium on PODS*, pages 221-230, 1988.
- [24] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceeding of the 5th Int'l. Conf. on Logic Programming Symposium*, pages 1070-1080, 1988.

- [25] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proceeding of the 7th Int'l. Conf. of Logic Programming*, pages 579–597, 1990.
- [26] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databaes. *New Generation Computing*, 9(3-4):365–385, 1991.
- [27] M.L. Ginsberg, editor. *Readings in nonmonotonic reasoning*. Morgan Kaufmann Publishers, Los Altos, Ca 94022, 1987.
- [28] Y. Hu and L.Y. Yuan. Extended well-founded model semantics for general logic programs. In *Proc. of 8th International Conference of Logic Programming*, 1991.
- [29] K. Konolige. On the relation between default and autoepistemic logic. *Artificial Intelligence*, 35:343–382, 1988.
- [30] R.A. Kowalski. Predicate logic as a programming language. In *IFIP 74*, pages 569–574, 1974.
- [31] R.A. Kowalski and F. Sadri. Logic programs with exceptions. In *Proceeding of the 7th Int'l. Conf. of Logic Programming*, pages 598–613, 1990.
- [32] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artifitial Intelligence*, 44(1):167–207, 1990.
- [33] K. Kunen. Negation in logic programming. *The Journal of Logic Programming*, 4(4):289–308, 1987.
- [34] K. Kunen. Signed data dependencies in logic programs. *The Journal of Logic Programming*, 7(3):192–230, 1989.
- [35] L. Li and J. You. Abduction-based semantics of logic programming. personal communication, Janurary 1992.
- [36] L. Li and J.H. You. Making default inferences from logic programs. *Computational Intellugence*, 7(3):142–153, 1991.
- [37] V. Lifschitz. Computing circumscription. In *IJCAI-85*, pages 121–127, 1985.
- [38] V. Lifschitz. Pointwise circumscription. In *Proc. of AAAI-86*, pages 406–410, 1986.
- [39] V. Lifschitz. Declarative semantics of logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 177–192. Morgan Kaufmann Publisher, Los Altos, CA, 1988.
- [40] V. Lifschitz. Extended logic programs as autoepistemic theories. In *Proc. of 10th International Conference of Logic Programming*, pages 101–114, 1993.

- [41] D. Lin. *Obvious Abduction*. PhD thesis, Department of Computing Science. University of Alberta, 1992.
- [42] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.
- [43] M.J. Maher. Equivalence of logic programming. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 627-658. Morgan Kaufmann Publisher, Los Altos, CA, 1988.
- [44] D. Makinson. General theory of cumulative inference. In *Proc. of 2nd Int'l. workshop on nonmonotonic reasoning*, pages 1-18, June 1988.
- [45] S. Manchanda. Declarative expression of deductive database updates. In *ACM 8th Symposium on PODS*, pages 93-100, 1989.
- [46] S. Manchanda and D.S. Warren. A logic-based language for database update. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 363-394. Morgan Kaufmann Publisher, Los Altos, CA, 1988.
- [47] Z. Manna and A. Shamir. The theoretical aspect of the optimal fixed point. *SIAM J. Comput.*, 5:414-426, 1976.
- [48] Z. Manna and A. Shamir. The optimal approach to recursive programs. *Comm. ACM*, 20:824-831, 1977.
- [49] V.W. Marek and M. Truszczynski. Reflexive autoepistemic logic for logic programming. In *Proc. of 10th International Conference of Logic Programming*, pages 115-125, 1993.
- [50] W. Marek and M. Truszczynski. Stable semantics for logic programs and default theories. In *Proc. of the North American Conf. on Logic Programming*, 1989.
- [51] J. McCarthy. Epistemological problem in artificial intelligence. In *Proc. of Fifth IJCAI*, pages 1038-1044, 1977.
- [52] J. McCarthy. Circumscription-a form of non-monotonic reasoning. *Artificial Intelligence*, 13:295-323, 1980.
- [53] J. McCarthy. Application of circumscription to formalizing commonsense knowledge. In *AAAI Workshop on non-monotonic reasoning*, pages 295-323, 1984.
- [54] J. McCarthy and P. Hayse. Some philosophical problems from the standpoint of artifitial intelligence. *Machine Intelligence*, 28:89-116, 1969.
- [55] D. McDermott and J. Doyle. Non-monotonic logic i. *Artifitial Intelligence*, 13:41-72, 1980.

- [56] D. McDermott and J. Doyle. Non-monotonic logic ii: Non-monotonic model theories. *JACM*, 29(1):33-57, 1982.
- [57] E. Mendelson. *Introduction to Mathematical Logic*. D. Van Nostrand Company, Toronto, Canada, 1964.
- [58] J. Minker. On indefinite databases and the closed world assumption. In someone, editor, *Later*. Springer-Verlag, 1982.
- [59] J. Minker, editor. *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann Publisher, Los Altos, CA., 1988.
- [60] M. Minsky. A framework for representing knowledge. In P. Winston, editor, *The Psychology of Compiler Vision*, pages 211-277. McGraw-Hill, New York, 1975.
- [61] R.C. Moore. Semantical considerations on non-monotonic logic. *Artificial Intelligence*, 25, 1985.
- [62] A. Naqvi and R. Krishnamurthy. Database updates in logic programming. In *ACM 7th Symposium on PODS*, pages 251-262, 1988.
- [63] L.M. Pereira, J.J. Alfreres, and J.N. Aparicio. Adding closed world assumption to well founded semantics. In *Conf. Proc. of Fifth Generation Computer Systems.*, pages 562-569, June 1992.
- [64] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27-47, 1988.
- [65] D. Poole and R. Goebel. Gracefully adding negation and disjunction to prolog. In *Proceeding of the 3rd Int'l. Conf. of Logic Programming*, pages 635-641, 1986.
- [66] D. Poole and R. Goebel. Theorist: A logical reasoning system for defaults and diagnosis. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 331-352. Springer-Verlag, 1987.
- [67] T. Przymusiński. On the declarative and procedural semantics of logic programs. Technical report, University of Texas at El Paso, 1987.
- [68] T. Przymusiński. Non-monotonic reasoning vs. logic programming: a new perspective. In D. Partridge and Y. Wilks, editors, *Formal Foundations of Artificial Intelligence*. Cambridge University Press, London, 1988.
- [69] T. Przymusiński. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193-216. Morgan Kaufmann, Los Altos, CA, 1988.

- [70] T. Przymusiński. On the relationship between logic programming and non-monotonic reasoning. In *Proc. of AAAI-88*, pages 444–448, 1988.
- [71] T. Przymusiński. Every logic program has a natural stratification and an iterated fixed point model. In *ACM 8th Symposium on PODS*, pages 11–21, 1989.
- [72] T. Przymusiński. Three-valued formalizations of non-monotonic reasoning and logic programming. In *Proc. of the 1st Int'l. Conf. on Principle of Knowledge Representation and Reasoning*, pages 341–348, 1989.
- [73] T. Przymusiński. Extended stable semantics for normal and disjunctive programs. In *Proc. of the 7th Logic Programming Symposium*, pages 459–477, 1990.
- [74] T. Przymusiński. Autoepistemic logic and logic programming. personal communication, 1991.
- [75] R. Reiter. On closed world databases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 55–67. Plenum Press, New York, 1978.
- [76] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 32:81–132, 1980.
- [77] R. Reiter. Nonmonotonic reasoning. *Annual Reviews of Computer Science*, 2:147–187, 1987.
- [78] N. Rescher. *Many-valued Logic*. McGraw-Hill Inc, 1969.
- [79] K. Ross. A procedural semantics for well-founded negation in logic programs. In *ACM 8th Symposium on PODS*, pages 22–33, 1989.
- [80] J.S. Schlipf. Formalizing a logic for logic programming. In *Intl. Symp. on Artificial Intelligence and Math*, Jan. 1990.
- [81] G.F. Schwarz. Autoepistemic logic of knowledge. In *Proc. of 1st Int'l. Workshop on Logic Programming and Non-monotonic Reasoning*, pages 260–274, 1991.
- [82] J.C. Shepherdson. Negation as failure, ii. *Journal of Logic Programming*, 2(3):185–202, 1985.
- [83] J.C. Shepherdson. Negation in logic programming. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 19–88. Morgan Kaufmann Publisher, Los Altos, CA, 1988.
- [84] J.R. Shoenfield. *Mathematical Logic*. Addison-Wesley Publishing Company, 1967.
- [85] M.E. Szabo, editor. *The Collected Papers of Gerhard Gentzen*. North-Holland, Amsterdam, 1969.

- [86] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5:285–309, 1955.
- [87] A. Tarski. *Logic, Semantics, Metamathematics, Papers from 1923-1938*. Hackett Publishing Company, Indianapolis, Indiana, 1983.
- [88] H.C. Wasserman and Z. Shen. Locality principle and completeness results. personal communication, February 1992.
- [89] M. Winslett. A model-theoretic approach to updating logical databases. In *ACM 5th Symposium on PODS*, pages 224–234, 1986.
- [90] M. Winslett. A framework for comparison of update semantics. In *ACM 7th Symposium on PODS*, pages 315–324, 1988.
- [91] M. Winslett. Reasoning about action using a possible model approach. In *Proc. of the 7th National Conf. on Artificial Intelligence*, pages 89–93, 1988.
- [92] J. You and L. Li. Answer set semantics for logic programming and default reasoning. personal communication, May 1992.
- [93] J.H. You and L.Y. Yuan. Three-valued formalization of logic programming: Is it needed? In *ACM 9th Symposium on PODS*, 1990.
- [94] J.H. You and L.Y. Yuan. A new circumscription method for stable model semantics. personal communication, 1991.
- [95] J.H. You and L.Y. Yuan. On the equivalence of semantics for normal logic programs. In *Proc. of Workshop on logic programming and incomplete information*, pages 161–176, 1993.
- [96] L.Y. Yuan. Unfounded atoms and well-founded beliefs of logic programs with negation. personal communication, April 1992.
- [97] L.Y. Yuan. Autoepistemic logic of first order and its expressive power. personal communication, June 1993.

Index

- $\preceq$ 21
- $\models_{\mathcal{L}}$ 20
- L_m 17
- δ_{Π} 34
- Δ_{Π} 34
- $\oplus$ 35
- $\vdash_{Horn}$ 56
- $\vdash_{Case}$ 56
- $\vdash_{L_m}$ 58
- $\Psi_{\Pi}(H)$ 48
- 2-valued interpretation 19
- 3-valued stable model 76 77
- 3-valued stable-by-case model 76
- alphabet 13
- answer set 61
 - GL-answer set 95
 - L_m -well-founded answer set 61
 - L_m -saturated answer set 61
 - L_m -prerable answer set 61
 - L_m -stable answer set 61
- argumentation semantics 98
- atom 14
 - ground atom 20
- bound 15
- complete lattice 43
- complete semilattice 44
- consistent 19
- continuous mapping 44
- cumulative monotone 118
- cumulativity 115 116
- directed set 44
- existential closure 15
- explanation 32
- extension 32
- first order language 14
- free 15
- fixed point 43
 - least fixed point 43
 - greatest fixed point 43
- GL-transformation 79
 - extended GL-transformation 75
- general program 25
 - normal program 25
 - extended program 25
 - definite program 25
 - disjunctive program 51
- Herbrand base 21
- Herbrand instantiation 23 26
- Herbrand universe 21
- hypothesis tuple 34
- inconsistent 19
- instance of a formula 23
- instantiated formula 23
- interpretation 16
 - Herbrand interpretation 21
 - partial interpretation 27
 - total interpretation 27
- literal 15
 - positive literal 15
 - negative literal 15
- logical consequence 20
- L_m -sound 60
- $\mathcal{L}$ -admissible 39 52 53
- $\mathcal{L}$ -false-assumable 38 59

- $\mathcal{L}$ -founded 39
 - positively $\mathcal{L}$ -founded 38 59
 - negatively $\mathcal{L}$ -founded 39
- $\mathcal{L}$ -noncontradictory 37 58
- $\mathcal{L}$ -saturated 40 52 53
 - soundly $\mathcal{L}$ -saturated 60
 - $\mathcal{L}$ -saturated closure 48
- $\mathcal{L}$ -preferable 40 52 53
 - soundly $\mathcal{L}$ -preferable 60
- $\mathcal{L}$ -stable 40 52 53
 - soundly $\mathcal{L}$ -stable 60
- $\mathcal{L}$ -well-founded 50 52 53
 - soundly $\mathcal{L}$ -well-founded 60
- model 19
 - Herbrand model 21
- monotonic mapping 43
- ordinal power $\Phi \uparrow^\alpha$ 43
- oriented consequence 55
- partial model semantics 28
- partial order 42
- preferential extension semantics 72
- prenex form 22
- principle of locality 105 109
- principle of proof by cases 125
- query 29
- ratinal monotone 118
- ratinality 115 116
- relevant program 104
- relevant set 104
- satisfiable 19
- scenario 32
- semantical instance 51
- sentence 15
- substitution 23
 - answer substitution 29
 - correct answer substitution 29
- stable extension 82
- stable set 80
- stable-by-case model 80
- supraclassicality 123
- term 14
 - ground term 20
- universal closure 15
- unsatisfiable 19
- valid 19
- variable assignment 16
- well-founded model 78
- well-founded-by-case model 79